uc3m | Universidad Carlos III de Madrid

Computer Engineering
(2017-2018)

Trabajo Fin de Grado

Incremental Learning for Volcano Monitoring

Francisco Javier García Barrero

Tutor

Agapito Ledezma Espino

5th of October 2018, classroom 1.2.C16

INDEX

FIGURE INDEX

TABLE INDEX

1. ABSTRACT

This document studies the creation of a computer program written in Java programming language to classify seismic movements occurred between 2015 and 2016 collected from volcano Puracé in Colombia, with the inclusion in this system of Data Mining application MOA by the University of Waikato.

It is offered an initial study of aspects to take into consideration for the problem, presenting for a more profound analysis of the questions from a technological point of view, various suggestions and ways of confronting it.

By performing a series of tests using surveillance data from volcano Puracé and applying modifications to these data, it's derived the accuracy of Data Mining algorithms and necessary processes to produce a base structure for real-time applications related to seismic movement analysis. The analysis of the data is also performed, along with derivation of models of prediction using the program built for this project.

2. ACKNOWLEDGEMENTS

First, I want to thank University Carlos III of Madrid for giving me the instruments to become a better professional, and in some ways, a more adult person than the one who entered this institution.

I want to give thanks to Agapito Ismael Ledezma for providing me the opportunity to make this work and helping me these months of doubts and effort. Also, I want to thank María Paz Sesmero Llorente for helping me setting up the experiments in the servers of this university.

Lastly, I want to thank my family, which has given me support during all these years.

3. INTRODUCTION

Volcanic events can create a vast amount of destruction and end the lives of many, sometimes without any kind of signal from which experts can predict a catastrophe. An example of this fact is the eruption of mount "Ontake" in Japan on the 27th of September in 2014 [1]. The only way to find methods to avoid damage from this kind of events in the future (if there's any likelihood of the existence of these methods in the future) is through extensive study. There is still much work to be done to be able to prevent these kinds of events and many techniques to be discovered, and this work is based on that premise.

Before being capable of prevention of these events we need to be able to classify them correctly given a series of parameters. However, the correct number of parameters, data structures and methods to use to accurately classify any given event with only a small part of its data are still unknown. Still, this doesn't mean there aren't ways to predict to some extent these events.

Currently, the instrument used for capturing the movements of ground produced by seismic events is the seismometer. This tool produces a graph (or seismogram) containing the amount and direction of movement of ground in three perpendicular axes: one vertical and two horizontal. These signals are measured as accurately as possible, meaning it's necessary to save the value measured at multiple points in time, as close in between as possible.

For this work the author will use data taken from seismometers placed around the volcano Puracé, situated in Colombia. The data taken has been classified by experts depending on the seismic event to which each part of the data corresponds to. The plan is to search for a mechanism to classify this kind of data with the use of machine learning algorithms and find relations between different parts of data; relations that may be hidden at plain sight and unknown in the field of volcanology.

Regulations related to the nature of the data used in the work and the processes involved in obtaining this data are identified in the section 3.5 (Legal Framework).

3.1. DESCRIPTION OF THE PROBLEM

Close to the volcano Puracé in Colombia, a set of seismometers, each in a different location, take real-time measurements of the movements detected from ground in three perpendicular axes. These measurements are produced synchronously after each 10 milliseconds, and stored in a database. At a later time, volcanology experts infer seismic events occurred by analyzing previous data on the whole. For each seismic event, these experts derive information, which includes: class of seismic event, starting date, finishing date, amplitude, frequency... This data is also saved into the same database as before [2].

The problem presented in this work is the following: can we correctly classify seismic events in real-time with the least amount of information possible?

The message “least amount of information” has a dual meaning here. Suppose we join data from the measures taken by seismometers into a structure (for example, the measurements of two different seismometers at the same point in time).Generating the smallest structures of data from which accurately classify events and using the least amount of units of these structures possible would describe both denotations.

To aid in the process of classification, various algorithms will generate models from the units of data structures read by or “feed” to them, by analyzing and saving characteristics of this data for classification. These models will vary in accuracy and time required to classify data.

After finishing creating the models from a limited amount of data, testing will be performed to compute the capacity of the models to classify data not previously used, but disallowing saving any new characteristics that can be used to modify their knowledge. The accuracy of models in the tests will be the quality from which the author may be able to prove the existence of mechanisms to correctly classify seismic events in real-time.

3.2. MOTIVATIONS

The initial idea for the subject of this final project was choosing something more related to graphical engines, but I felt the knowledge gained through my studies about the topic of Data Mining wasn’t enough for me, so I wanted to compensate for that by being able to solve more complex problems than the ones presented in class, or at the very least, problems with a real data domain.

Another of the reasons behind this choice was: which of the topics proposed by the university would be more useful to spend time on solving? (Not only more useful for us, but also for others). Comparing the creation of light applications serving as ways of enjoyment to the capability of saving lives by predicting an eruption (at least, in case the results of this work are optimal) transforms the question into an easy choice, and so, this line of reasoning ended up pushing us to pick the later.

It must be noted that being able to have access to a great amount of data of seismic measurements is also a rare opportunity, since in other cases it would be harder to find this quantity and, most importantly, expensive to access this kind of data.

3.3. OBJECTIVES

The objective proposed for this project was an initial study of seismic measurements from volcano Puracé in order to predict the type of seismic event occurring at any point in time. When the first models for this were produced, their results were not as good as expected, pushing us to search for ways to find better results. New lines of thinking arose, and from multiple modification schemes for the data three different approaches were taken, which will discussed in a later section. The author’s intention is to find the results from these approaches so that it serves as a point of view for future works in the fields of data mining and volcanology, which could allow knowing which operations would be useful to perform over seismic data to predict its class, and which wouldn’t.

As the project evolved so did the objectives, turning a process of analysis into one of automatization. And so, this document not only describes the study of the data, but also the basics for the creation of a Java language based data mining application with multiple functions: automating database petitions, preprocessing of data, training of data mining algorithms, and the loading and saving of models and results.

Although not strictly necessary, it's also included the creation of a basic graphical user interface or GUI to speed up experimentation and allow for any person with access to the application to make their own personalized experiments (with some of the parameters already set). The author hopes this document can also help other programmers of data mining applications with common and not so common problems they may find in the creation process.

Finally, it is expected for the results of the analysis performed to serve as instrument to open new lines of investigation on the subject of volcanology.

3.4. STRUCTURE OF THE DOCUMENT

The structure of next parts of the document is the following:

-State of the art, starting with previous projects, theories and data discovered from which this work has benefited, followed by the definition of core concepts and the methodology used to bring the application to completion.

-The development, with the analysis of data, modification types used for files and explanations about the core of the application from three points of view: design, architecture and implementation.

-The experimentation, with the tests performed over the data of volcano Puracé, discussing models created, processes followed to obtain the experiments and discussions about the results.

-Conclusions about the project as a whole and usefulness of the work produced.

-Future works proposed to continue investigating the different subjects of this project in order to find new discoveries.

-The budget necessary for this project to be completed. It can be used as a guide for similar projects to compute costs as well.

-The socioeconomic impact that this work may produce.

-A bibliography including references for the works that served this project as foundation, necessary tools and other bits of information.

3.5. LEGAL FRAMEWORK

The data used for this work belong to the Geological Observatory of Colombia, which transferred to CAOS group from University Carlos III of Madrid. The website giving public access to the data stored by this university was published in the document "Monitoring of

Vulcano Puracé through seismic signals: Description of a real dataset" [2]. The data was made accessible for the public with research purposes.

The data referred here is beyond the scope of protection given by Constitutional Law 15/1999 from 13th of December, of Personal Data Protection, given that the information used doesn't fit the concept of Personal Data which defines the Law in the section a) from its article 3 ("Any information concerning physical persons identified or identifiable"), by which, if the data extracted from the indicated source is correctly referenced to the same source, the author won't be incurring in any responsibilities by violation of arrangements included in the Spanish legislation.

On the other hand, the Law considers adequate and, by such, compatible with the ends by which the data was recovered, the later treatment of the very same data with historical, statistical or scientific purposes (article 4.2 from Constitutional Law 15/1999), which fits completely with the purposes of this scientific study.

Related to the consent of affected parties (even though in this case there are none, since the data doesn't concern any physical person), section 2 from article 6 of Constitutional Law stablishes that "consent won't be necessary (…) when the data is in sources accessible to the public and its treatment is necessary for the satisfaction of legitimate interests pursued (…), given that it doesn't violate the fundamental rights and liberties of the interested party"

In conclusion, the treatment of obtained information and the later usage with scientific purposes doesn't infringe the regulations imposed by current legislation, given that the nature of the data doesn't concern any physical person, and the data was obtained from sources of public access and referenced to the same sources.

4. STATE OF THE ART

In this chapter we will explore the necessary information, mechanisms and methodologies that were necessary for the development of the application and the subsequent experiments produced with it before its start. The concepts presented here will define the context of the rest of the work, and it may also serve as starting point for projects of the same matters as this one.

4.1. PREVIOUS STUDIES

In 2016, a paper with title "Monitoring of Vulcano Puracé through seismic signals: Description of a real dataset" was published. This paper described the creation and structure of a database containing seismological data from volcano Puracé in Colombia. The data was taken from different seismometers located in the areas close to the volcano and divided by instances or measures at an exact point in time. These instances were later classified by experts with the seismic event to which they belonged. In the paper it is explained that this dataset is published in order to allow any researcher to discover the possibility of using machine learning algorithms to create a real-time application able of classifying correctly this kind of data. And so, this database will be the one used for this study [2]. It must be noted the tutor for this work was among the authors of this paper, meaning that the author could solve any questions related to it without much effort.

A year after its publication, the student María de las Mercedes Crespo Jiménez from this university published a final project titled "Classification of Seismic Waves with Data Mining techniques", based on the objective proposed in previous paper [3]. This project proved the efficiency of algorithms C4.5, Hoeffding Tree and Bayesian Networks among others including clustering algorithms with a subset of the database. It must be stated that, after working on this project, the author believes the amount of instances used in this study to be insufficient (and we will also find that using the whole database will be insufficient in some terms). After training algorithms like Hoeffding Tree with subsets ten times larger and a hundred times larger, the difference in accuracy was found to be close to 15% and 30% lower than expected in the study respectively, which still proves viability for some of the statements made about the accuracy of algorithms, but not to the same extent. Nevertheless, the analysis of seismic events included in the document was very well arranged and of great help to understand the kind of problems the author faced here.

The database used had been upgraded after the publication of last study, including files that weren't previously available, giving us higher odds of finding substantial results.

4.2. SEISMOLOGY

Seismology can be defined as the study of seismic events, the propagation of seismic waves and their sources, mostly based on processes related to collection of data and classification and prediction of events [4].

The effects of these events can be devastating, reason why in ancient times this field was of interest for scientists, philosophers and scholars, although its creation souldn't be considered until 1889, year in which the first prototype of a machine to measure movements of the ground was invented: the seismograph or seismometer [5].

Seismometers have evolved over time, for example, upgrading the accuracy of the measures taken or transforming from analogical to digital systems [5]. The case studied will include measurements of movement of ground from three perpendicular axes, with each measurement taken every 10 milliseconds. Although these devices may seem reliable, we will find seismometers can sometimes still produce errors through the course of this work.

The author will use the categorization of seismic events made by experts, of data retrieved by various seismometers around volcano Puracé. The main objective will consist in assisting data categorization, and if possible, prediction of events to create models which will adjust to the evaluation provided by experts using different mechanisms.

The data which will be analyzed can be classified into 18 different classes, each for a different type of seismic event. From this set only 14 will be included in the data from this work. It must be stated the class "NULL" included in one of these 14 classes is a construct of the data that can't fit into any other class. These classes and their corresponding codification in the database are the following:

-Volcano-tectonic (VT): generated by fracturing of rocks adjacent to magma reservoirs, usually associated with ascent or intrusion of magma [2].

-Long period (LP): generated by gas or fluid material within volcanic conduits. Characterized by frequencies which are smaller than 5Hz [2].

-Screw (TO): long period events of monochromatic frequency spectrums and a soft decay found in the image produced [2].

-Local tectonic (TL): originated by active faults close to the volcano [2].

-Regional earthquakes (RE): events which origin is closer to that of distant seisms but farther to that of local tectonic events.

-Distant seism (DS): used for seisms originated in other countries or at very large distances from the seismometer.

-Tremor (TR): events of origins similar to those of long period events. They are sometimes characteristic of volcanic eruption. Its duration is very variable [2].

-Mine explosion (EM): produced by explosives in an underground area. It is safe to assume the effects of explosives considered of "high order" to last for longer periods of time than other kinds. While it would be impossible to predict this kind of events since they are produced by the action of man, most can be predicted by other means (as asking the people in charge of those explosives). It is still interesting to classify these events to differentiate them.

-Surface activity (SU): produced by events on the surface not classifiable as those produced from ice, lightning or avalanche given they are considered within their own

class. One example of this kind of events would be the effects produced by a truck moving close to a seismometer generating vibrations. It is difficult to predict, but interesting for classification to avoid confusion with other events.

-Non-classifiable (NC): seisms without classification but clearly distinguishable from others events or lack of events.

-Calibration (CA): occurring when calibrating seismometers. Unpredictable by analytical methods, but predictable by other means (such as asking the regional geological institution). It is still interesting for classification to differentiate this kind of events and avoid confusion.

-Lightning (RY): produced by lightning going from cloud to earth. The area of a volcano is usually more prone to lightning given the charge produced frictions underground, and in the case of active volcanoes, because of rock fragments colliding which generate static electricity [6].

-Low frequency (BF): events of low frequency with different image and characteristics than those of long period or screw type.

-Undetermined (ND): seisms for which choosing a class may produce misclassification because of the complexity of the image produced.

-Null class (NULL): lack of seisms or events with little to no damaging effects produced.

The classes not included inside the data used and their corresponding codification in the database are the following:

-Hybrid seism (HB)

-Cryoseism (HI)

-Explosion (EX)

-Avalanche seism (AV)

Each of these classes, with the exception of "NULL" and "NC" classes, produce seismograms (or images drawn by the seismometer) that share characteristics such as shape of the wave and/or length among others. If these characteristics are compared between classes, each class has their own distinctions.

4.3. DATA MINING

Data mining was the name given to the process that encapsulates extracting from a set of data different patterns through the use of algorithms. It was commonly misused as a more general term substituting that of Knowledge Discovery in Databases or KDD, and although data mining is only one of the steps of this procedure, its meaning has evolved over time to become that of Knowledge Discovery in Databases. The reason for this may also reside in the creation of

standards around this substitution like CRISP-DM, which stands for Cross-industry standard process for data mining [7] [8].

Today, data mining as described by CRISP-DM standard, defines the union of processes of analysis of business, analysis of data, data preprocessing, generation of models, evaluation of results and deployment; all serving as method to discover new knowledge from sets of data. There are multiple methodologies involving different steps and number of steps, all related to steps from the standard mentioned, but changed as suited for the problems each system has to resolve [8].

The processes included in this work resemble those of the standard CRISP-DM, only with a few differences, explained in section 4.5 (Methodology).

The algorithms used for the experiments are implemented in the MOA library included in section 5.4 (Tools).

4.4. INCREMENTAL LEARNING

Incremental Learning serves as the technique used to extend the knowledge of models created by multiple machine learning algorithms, usually an option for problems for which the data is collected over time and the whole set of information to analyze isn't available all at once, or the amount of data that composes it is too big.

The nature of problems for which incremental learning serves purpose include data sets where changes in the distribution of data occur over time and/or where noise is included in the data. [9]

The case of a real-time application to categorize seismic events fits this description, since between the tools used appear machine learning algorithms capable of the extension of their knowledge after a model has been generated (note that not all kinds of machine learning algorithms include this capability), the set of data is big enough it becomes necessary to process it by parts, and more data than the available will be needed to be classified depending on human feedback. [10] [9]

4.5. METHODOLOGY

As of 2014 the most common methodology for data mining projects is CRISP-DM, followed by created-from-scratch methodologies used only once for specific tasks (which translates to a personal methodology) [11]. In this case, the methodology used could be close to that of CRISP-DM, but with some differences. To understand why, let's explain first CRISP-DM.

CRISP-DM methodology splits the process of a data mining project into 6 differentiated phases:

> -Business understanding: determining the initial objectives, the situation in terms of resources, information, costs and requirements, planning the scheme of the project as a whole and selecting tools and techniques.

-Data understanding: all operations related to collection, description and analysis of data.

-Data preparation: selection, cleaning and other operations to modify the data depending on the specifications of the task.

-Modeling: selection, building and testing of models produced by algorithms.

-Evaluation: review of results and process, and exploration of next steps to follow.

-Deployment: planning of installation, monitoring and maintenance of the solution and creation of documentation related to the process as a whole.

To go from one phase to other of a project and consider the methodology to be CRISP-DM [12], we need to follow next diagram:

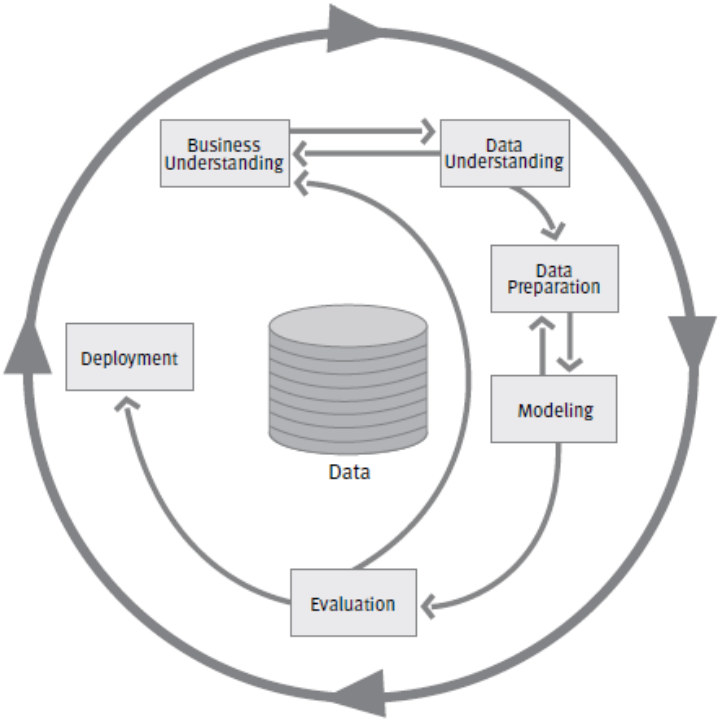

Figure 1: Flow diagram of CRISP-DM methodology [12]

Although the author followed a similar process for this work, the multiple approaches taken and developed at different speeds made possible going from the phase of data preparation to the phase of evaluation, among other possible phase changes that aren't included into the flow of CRISP-DM. This the reason why the methodology used is considered to be "the author's own methodology", selected specifically for this project.

5. DEVELOPMENT

This chapter offers the instructions to build a system for the analysis of data. While this case focuses on the task at hand of volcanology, it can be useful for the developing of other systems requiring data mining techniques. The system will be named after the volcano analyzed for the study: Purace. Note that when referring to the application, the name won't include an acute accent such as the one of the volcano.

5.1. APPROACHES

Given the rich dataset from the previous paper about volcano Puracé [2], we have the opportunity to search for relations between three kinds of data: type of seismic movement, the measures of motion from the ground computed in three perpendicular axes (given by the seismometers spread around the volcano) and the time at which a movement occurs. This kind of relations may be too complex to be appreciated at first glance, even more if it has to be derived new knowledge from millions of instances of data; this is where data mining algorithms become powerful tools.

The main approach will consist in finding the capacity of data mining algorithms to correctly classify the type of seismic movement given the rest of the variables using data from the database. For this it is custom to assume as correct all given instances measured as any kind of seismic movement, and hide this value to the algorithm until it gives us its prediction, which then is compared to the actual value of the instance to compute accuracy rates and fed to the algorithm, so it can modify its internal values if necessary. The author will also compute the time expended by the algorithms used to learn from instances, which if low enough could allow for the creation of a real time application to surveil the state of not only Puracé, but any volcano (given, of course, data structures close to ours).

We can also imagine that if the main approach gives successful results, there could be unknown functions that define ways to, given this dataset, predict the next seismic movement that will be produced from any point in time without taking into account the exact moment at which it will happen, forming the second approach. From this estimation it could also be tried to find results that are produced from using bigger spaces of time between predictions or a higher amount of movements of difference between prediction and occurrence of any event.

The last approach consists in finding the existence of an implicit relation between the waveforms of all types of seismic movements; one of which nature would allow us to split any kind of waveforms into equal time portions, no matter the length of the wave (or, which is the same, the period of the movement). If proven right, we could predict by some margin the time at which a movement reaches decay (depending on the type) or the duration of the movement. Even if proven wrong, this kind of data split could also be studied event type by event type, analyzing each separately and generating other results that could be of high accuracy in some cases, given the step down in the complexity of the task from a wider perspective to a focused analysis.

5.2. ANALYSIS OF DATA

From the beginning of the project, the author was allowed access to the database from the Colombian Geological Survey (SGC) [2]. This database includes directories divided by year, and inside those directories, folders divided by month and type of file. Inside these folders, there are files for each day of the corresponding month, numerically named depending on the number of days the year has (from 1 to 365 or from 1 to 366 in the case of leap years). These files are divided in two types: class data (specifying the starting and finishing date of seismic movements and their class, among other data) and instance data (defining measures taken from distinct locations around the volcano Puracé and date at which they were taken).

In this case, the existing data comprises years 2015 and 2016, but since the data is not uploaded in full, some folders don't exist, which becomes a problem since it's impossible to use partial data (this is, the lack of classification of instances or lack of instances) for this classification task. A look into its contents tells us it is available all necessary data from days 1st of August 2015 onward to the 31st of January of 2016 (including this day). In a real case it will be likely to have large amounts of data inside a regimented time frame, so it will be convenient to use this disposition for experiments.

It will be now described the two kinds of files included in the database:

5.2.1. Class instances

Each file begins with an ordered definition in Spanish of each of the values found in each class instance, which are:

- Class of seismic movement.
- Starting date of *P-Wave* of the seismic movement (in *long* format)
- Starting date of seismic movement
- Starting date of *Coda* of the seismic movement (in *long* format)
- Finishing date of seismic movement
- Starting date of *S-Wave* of the seismic movement (in *long* format)
- Duration of seismic movement in seconds (rounded up)
- Amplitude of seismic movement
- Period of seismic movement in seconds
- Frequency of seismic movement in hertz

Next follows an example of the first two lines of the class file *214.txt*:

```
1  tie_codigo, lec_p,fecha inicio(P),lec_coda,fecha fin (coda), lec_s, duración, lec_amplcuen, lec_periodo,lec_frencuencia
2  LP,1438474550.1199999,2015-08-01 19:15:50.12-05,1438474565.3399999,2015-08-01 19:16:05.34-05,0,15,1184,0.12,8.2100000000000009
```

Figure 2: Beginning of the class file "214.txt" corresponding to the 2nd of August 2015. File included in database from Colombian Geological Survey [2]

The *long* format of three of the variables indicates a specific date, by using a value measuring the number of milliseconds passed from the 1st of January of 1970 at 00:00:00.000 hours. This kind of values will be necessary in order to operate a common domain for instances, used in

many other applications as standard, reason why the values of starting and finishing dates will also be transformed into this format.

For this project the only focus will be in using from class data files the attributes of class, starting date and finishing date of the seismic movement. Since this is an initial analysis of the problem, the author will use only fields that are believed to be measurable from each given instance alone. Other values contained inside class data files might be hard to measure correctly from each given instance since they obey computations that require multiple instances and/or knowing starting and finishing dates of a seismic event.

With this information, we could define any class instance with this simplified description:

Tie_codigo	**Fecha inicio(P)**	**Fecha fin(Coda)**
Class of seismic movement	Starting date	Finishing date

Table 1: Necessary attributes of a class instance

5.2.2. Data instances

Each file begins with an ordered definition in Spanish of each of the values found in all data instances. Since the contents of all variables will be used for the next steps, this time a table will be used to describe its values:

data	Date of instance measured to milliseconds
ABLO.x	X axis from seismometer ABL
ABLO.y	Y axis from seismometer ABL
ABLO	Z axis from seismometer ABL
CO2O.x	X axis from seismometer CO2
CO2O.y	Y axis from seismometer CO2
CO2O	Z axis from seismometer CO2
COBO.x	X axis from seismometer COB
COBO.y	Y axis from seismometer COB
COBO	Z axis from seismometer COB
LARO.x	X axis from seismometer LAR
LARO.y	Y axis from seismometer LAR
LARO	Z axis from seismometer LAR
SHAO.x	X axis from seismometer SHA
SHAO.y	Y axis from seismometer SHA
SHAO	Z axis from seismometer SHA

Table 2: Necessary attributes of a data instance

Table 2 also shows code names of the five seismometers from which measures were taken for this study: Agua Blanca (ABL), Cocuy 2 (CO2), Condor (COB), Lavas Rojas (LAR) and Shaka (SHA).

The measures of a sixth seismometer with codename “PIL” (Pilimbala) will be included in the original data of some instances, but in the modification process its data won’t be included for reasons explained in section 5.5 (Design).

If we open a data file, we will find an initial line indicating this structure, followed by one instance per each new line of data. Here is an example of the first two lines of the data file "day213.txt":

```
data,ABLO.x,ABLO.y,ABLO,CO2O.x,CO2O.y,CO2O,COBO.x,COBO.y,COBO,LARO.x,LARO.y,LARO,SHAO.x,SHAO.y,SHAO
2015-08-01T00:00:00.000005,-9559,-10155,-9943,-9725,-8811,-9339,-8670,-8710,-7719,-8316,-9711,-13354,15230,-8875,-33560
```

Figure 3: Beginning of data file "day213.txt" corresponding to the 1st of August 2015. File included in database from Colombian Geological Survey [2]

If we compare sizes of files, most class instance files take less than 20KB of space, while data instance files occupy around 1GB of space. The reason behind this difference is that class instance files indicate seismic movements occurred through a day, while data instance files contain the values of all seismometers at each ten milliseconds of a day. A quick calculation gives us a number of 8.640.000 data instances per day (one for each ten milliseconds), deriving in 8.640.001 lines of text inside each data instance file.

If the reader obtains one of this data instance files or another file of size close to the one of this files and insists in opening it, consider checking section 5.4 (Tools) to avoid some problems related to heap space.

5.3. MODIFICATION TYPES

At the start of the project the author created various initial modifications of the data to get a handle of the tools and fix mistakes. In the end, a modification scheme for each of the approaches to prove their viability was determined. This section will explain the logic behind each of these modifications. Since there is one modification for each approach, they will be presented using the same order in which these were presented in the document.

5.3.1. First approach –Modification 6-

For the first approach, we must find ways to classify the event occurring at any point in time. By using the measures taken by seismometers and adding units of data which we may consider, we will give information which algorithms may use to differentiate behaviors of the waves produced in the ground as effect of those events.

Before going into detail, we will try to understand what kind of data we will be working with. A seismometer takes measures using three axes: two horizontal which are perpendicular between them and one vertical. Let's look at the representation of measures taken by one of the axis of a seismometer for a seismic event.

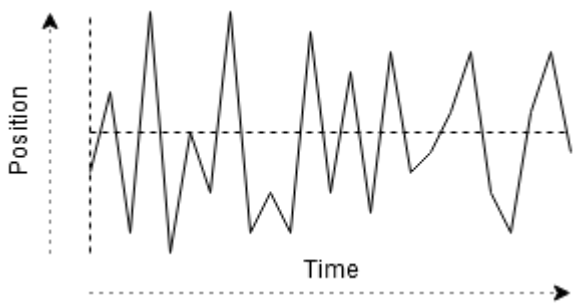

Figure 4: Representation example of the image of an axis of a seismometer

The measures taken at each point in time are united by a line, forming a picture that resembles figure 4. In this case, each of the measures will be divided by a time difference of 10 milliseconds. The position depends on the elastic waves produced by the movements of the ground.

Each of the instances of data given follow a more simple structure, saving only the data strictly necessary to create a representation like the one included in figure 4, which would only use the points of union of each of the lines. There are three axes for each seismometer and 5 seismometers in total. As explained before, each data instance contains then 15 values. If it was shown only one axis of a single seismometer, the data recovered could by represented by figure 5:

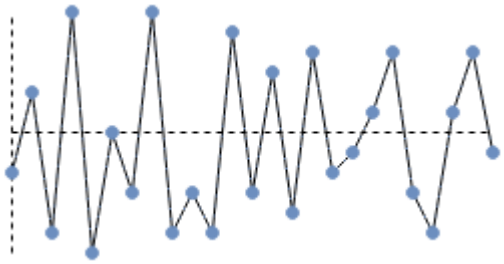

Figure 5: Representation example of part of the structure given by data instances using the image of an axis of a seimometer

If we intended to use the given format of data instances paired with the class to which each one belongs to, we would be analyzing a static image of each measure of a seismic event. This would mean for the algorithm that differentiating classes would be trivial and only depend on the position of each of the axis of a seismometer. In that case, the amount of movement would be unnecessary, which would also mean that reaching the same position with two different events shouldn't be possible in order to avoid mistakes between classifications of instances, which is not the case.

Prediction of events would be done then by comparing positions of seismometer axis with a range of values belonging each to its own class. The author believes the nature of this problem to be more complex.

With a powerful machine, we would be able to analyze the union of multiple seismometer positions forming each of the points that belong to the whole picture of a seismic event, but sadly for everyone, this is not possible yet (at least with the time and tools provided). Still, let's try to find out what can be predicted with a first step in this direction. It will be used not only the state of multiple seismometers, but the change produced from one state to the next, creating what could be described as a vector of movement with this state as starting point.

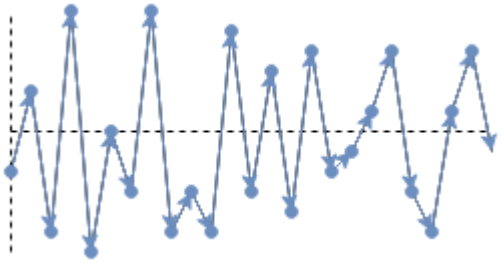

Figure 6: Representation example of part of the structure created for data instances using the image of an axis of a seismometer

Each point and arrow attached would be measures taken for one axis of a single seismometer. Each vector would be composed by one initial position (each of the points in figure 6) and one movement (each of the arrows in figure 6), both of course attached together. The structure of a modified data instance (without counting the class to which it belongs) would be composed by the union of vectors measured at the same point in time from the three axes of five seismometers, making use of 15 vectors, including then a total of 30 values (31 with the class to which each instance belongs to).

There is a slight disadvantage to the implementation given this structure. If we wanted to generate an instance from the last of the measures from a day, we would need the first measures taken from the next, and it would imply downloading (as we will see later) not only one, but two files of a size around 1GB before finishing the modifications of a whole day of data. To solve this one instance less than expected for each day of data will be used, meaning using 8.639.999 instances per each day of data instead of 8.640.000. From the big picture, it will only eliminate 184 instances out of the total possible of 1.589.760.000.

The structure of each data instance of the modification will be the following:

ABLO_x	X axis from seismometer ABL
ABLO_y	Y axis from seismometer ABL
ABLO_z	Z axis from seismometer ABL
CO2O_x	X axis from seismometer CO2
CO2O_y	Y axis from seismometer CO2
CO2O_z	Z axis from seismometer CO2
COBO_x	X axis from seismometer COB
COBO_y	Y axis from seismometer COB
COBO_z	Z axis from seismometer COB
LARO_x	X axis from seismometer LAR
LARO_y	Y axis from seismometer LAR
LARO_z	Z axis from seismometer LAR
SHAO_x	X axis from seismometer SHA
SHAO_y	Y axis from seismometer SHA
SHAO_z	Z axis from seismometer SHA
ABLO_x_move	Movement of X axis from seismometer ABL
ABLO_y_move	Movement of Y axis from seismometer ABL
ABLO_z_move	Movement of Z axis from seismometer ABL
CO2O_x_move	Movement of X axis from seismometer CO2
CO2O_y_move	Movement of Y axis from seismometer CO2
CO2O_z_move	Movement of Z axis from seismometer CO2
COBO_x_move	Movement of X axis from seismometer COB
COBO_y_move	Movement of Y axis from seismometer COB
COBO_z_move	Movement of Z axis from seismometer COB
LARO_x_move	Movement of X axis from seismometer LAR
LARO_y_move	Movement of Y axis from seismometer LAR
LARO_z_move	Movement of Z axis from seismometer LAR
SHAO_x_move	Movement of X axis from seismometer SHA
SHAO_y_move	Movement of Y axis from seismometer SHA
SHAO_z_move	Movement of Z axis from seismometer SHA
class	Seismic movement of the instance

Table 3: Necessary attributes of a data instance for files using modification type 6.

5.3.2. Second approach –Modification 7-

Even if we found out good results for the products of the first approach, succeeding in the second approach seems farfetched. Still, the author believes this line of thinking may be beneficial for future works, so analysis will be performed for the behavior of the best algorithms for last modification to the prediction of events without the limitation of knowing the exact point in time in which they occur. This can be translated to transforming the "NULL"

instances (which correspond to instances not belonging to a seismic event) by adding to them the class of the next event occurring. The values of the field added to instances which belong to other classes will be those of other classes, meaning that "VT" class instances will include now a new field with the value "VT" and so on. This is done to properly associate the modified "NULL" instances with seismic events as a preamble.

Using the structure of modification 6 it will only be added the following field at the end of each instance:

classPredicted	Next seismic movement of different class than the current event

Table 4: Necessary attributes of a data instance for files using modification type 7 in addition to the fields from table 3

5.3.3. Third approach –Modification 8-

This time all events will be split in 16 equal parts, except the ones included in the class "NULL".

In events that last for number of tens of milliseconds different than a multiple of 16, this will however generate unequal splits, but these splits will be divided as equal as possible.

With the third approach, it is expected to find a common logical structure shared by all seismic events that allows us to know (if the previous split is performed) which part of the 16 an instance belongs to.

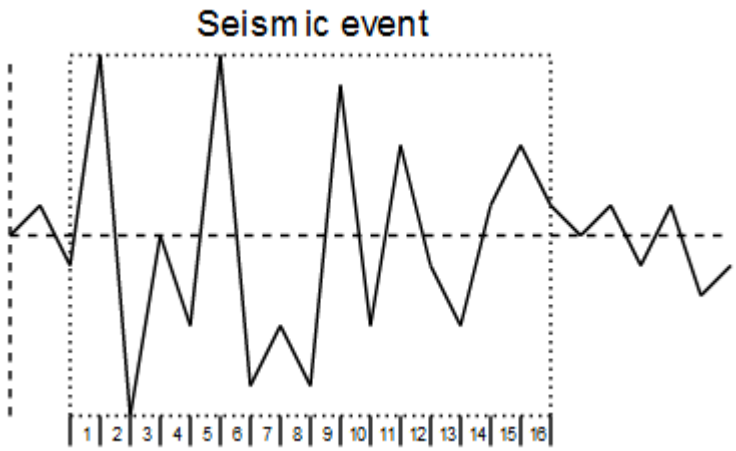

Figure 7: Representation example of the image of an axis of a seismometer, including the example of a seismic event divided in 16 equal parts

How could that be useful for us? The reader may wonder. Let's say we want to know how long a seismic event will take and we fortuitously found a way to find the split of any instance:

Knowing that between any two instances there ten milliseconds of difference, if a real-time application found that one instance has a value indicating the first split and the second one indicating the second split, we would easily approximate the length of the seismic movement to 160 milliseconds by rule of thumb.

Let's imagine this time the first instance belongs to the first split, but only starting the 401th instance we can start classifying the next as part of the second split. Again, a rule of thumb will

tell us the length of the seismic movement, being this time 1 minute 4 seconds. This time we only needed four seconds to correctly estimate a movement such length.

In a real event correctly measured, you could estimate the length of each event from the moment it starts, progressively until the arrival of an instance classified as second split.

Using the structure of modification 6 only the following field will be added at the end of each instance:

notePredicted	-1 in case of NULL class. From 0 to 15 otherwise, indicating the part where it belongs if splitting current seismic event into 16 equal parts

Table 5:Necessary attributes of a data instance for files using modification type 8 in addition to the fields from from table 3

So the reader may understand how this modification transforms the classes of instances, figure 8 was included. Each of the measures with the class -1 would be, as explained before, the ones corresponding to the NULL class. It is a custom in most programs to start counting calling the first of a series of objects with the number cero instead of one. This is the reason behind the chosen class value for seismic events.

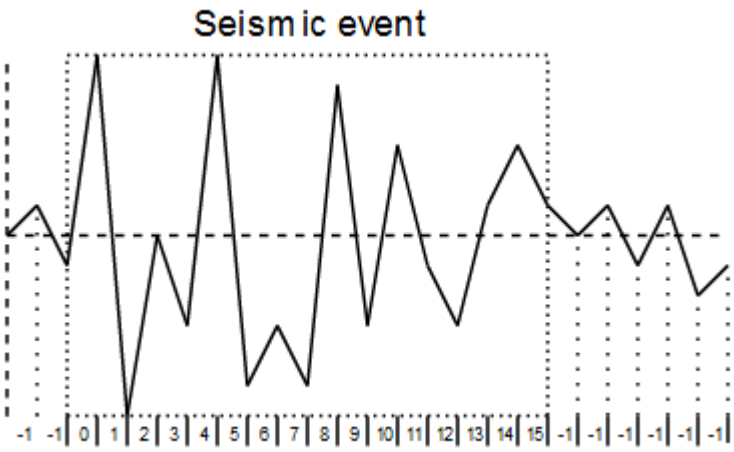

Figure 8: Representation example of the image of an axis of a seismometer, including a simplified example of a seismic event divided in 16 equal parts and the classification of instances given by modification type 8

Looking at the type of the seismic events found, as long as it isn't of "NULL" class, the value of instances for those classes will be comprised between zero and fifteen. It must be explained that the length of events may not be a multiple of 16, producing small errors in the process. An example of this is an event that lasts for 150 milliseconds and is measured by 15 instances. When the length of the event is larger than 160 milliseconds and also not a multiple of 16, the difference between an instance belonging to one part or another of the event may not be precise, producing misclassification. The number of instances misclassified for each event corresponds to the remainder of the total number of instances from the event divided by 16.

5.4. TOOLS

To proceed selecting the tools, we need to know which kind of operations we will have to handle. Note that there are some primordial tools which we won't be able to exclude from the process, like the database specified before [2] and an operative system. In this case the author will use Microsoft Windows 7, but most modern versions of Windows, GNU/Linux or even Apple Inc. operative systems will work as well, and in cases in which a platform exclusive tool is used, other suiting alternatives will be offered.

To produce the application, we will need tools depending on the objectives needed to complete; objectives which are the following:

- Downloading from the database the necessary data
- Opening and modifying raw text data of sizes close to 1GB
- Programming from scratch data mining algorithms or obtaining libraries that contain them
- Programming and testing the application
- Saving results and upgrading the visualization of data
- Displaying an interface for experiments

For each of these six activities, there is a tool that will provide the functions we need. Let's discuss in this order of objectives which ones will be useful and which will be selected.

For starters, a web browser will allow us to retrieve files from the database, but a question comes to mind: what could happen if we wanted to analyze the website of this database in order to automate this process? For this case, some options may be better than others… Which one should we pick? The author's answer is Mozilla Firefox, which contains very complete and easy to use web developer tools, but the author agrees that Google Chrome would be advisable as well.

The author hopes the reader followed the advice given in section 5.2 (Analysis of data), since this problem can be trickier than it looks. After obtaining the first data instance file various applications were tried to opening them, including the ones within the operative system, but doing so resulted in multiple errors referring to the impossibility of loading whole files into memory. Using the already selected web browser, the author was able to found Editpad Lite 7 [13], but this option may not be suitable for other operative systems. The author suggests though, Vim text editor [14] with LargeFile plugin [15] in these cases.

The tutor, Agapito Ismael Ledesma, suggested using Massive Online Analysis application from Waikato University. "**M**assive **O**nline **A**nalysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams. MOA includes a collection of offline and online methods as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, all with and without Naïve Bayes classifiers at the leaves" [16].
Using this tool is interesting from the point of view of future applications derived from this work, since the use of evolving data streams could produce systems that predict in real time

seismic events. It can be used as an application by itself or as a library and its code is written in Java programming language, solving portability problems to other operative systems. This however, will oblige us to use Java, making it the language in which the system will be written.

Now that we have the language, we can focus on finding a proper Integrated Development Environment or IDE to program and test the system with. University Carlos III of Madrid imparts Java programming classes using Eclipse IDE, and since the author is already well versed with this tool, it will be the best decision. Eclipse allows running and testing software, using multiple programming languages and it works in other platforms as well.

To fulfill the fifth objective the chosen operative system already has the answer for spreadsheet data: Microsoft Excel. This will allow using simple formats like *csv* and creating multiple kinds of graphs from data. However, it was mentioned that other alternatives would be offered, and for those using GNU/Linux or those who may have to spend money in order to acquire it, Excel may not be a good one. The author suggests LibreOffice Calc, which possesses some functions that Excel doesn't have and it's free.

The last objective can be completed with the help of a library for Java programming language called "Swing". This will allow us to display a window and fill it with buttons, images and other elements that may be required to draw on the screen.

With this the reader will be able to have all necessary tools to start creating a system for seismic movement classification. But tools itself are only the framework of this project, so next section will cover the design of the application.

5.5. DESIGN

In last section six compounds of operations to fulfill were described, and a good way to start this one is by asking ourselves, in which order should we tackle them? Depending on the abilities of the reader (in case he/she is the one programming the system) this choice may vary. Some people which work is involved with computer networks may prefer automating the petitions to database first for example. Anyhow, we will proceed with the order of class creation the author chose, overviewing the whole process but leaving the inner workings of the system as a whole for the next part. The format of problem/analysis/requirement will be followed for design guidelines when possible, hoping to make explanations more straight-forward and less dull for the reader.

On a side note, the order in which most of these guidelines for design were created isn't the one described here. As usual with data mining projects, multiple revisions of the analysis phase were made after tests, and many features implemented to adequate new problems arisen from experimentation.

5.5.1. Data operator

We start by downloading and analyzing files of the database, at least one of each type. The initial objective will be to modify instance data files to associate each instance with its corresponding class.

With the previous analysis the structure that encapsulates all files of each type was observed, making the task of operating these much more structured, but there's already a problem within files itself:

The size of instance data files makes it hard to operate them.

The reader may wonder if there is a way to split the data that is coherent and at the same time manageable. We know that these files always have the same number of lines (one for the name of the variables and 8.640.000 for the instances) and variable values, that this data is already ordered, and that it always belongs to a full day. These facts will help derive the first design rule:

-When operating with instance data files, first its data will be split into hours (360000 instances if a modification doesn't require data from more than one original instance to create a modified instance).

Class instance files come with their own share of problems:

Class instance data files correspond to data from half of the current day and half of the next day.

This unexpected problem will oblige us to fix class data files before merging its data with other files. The next decision will be:

-When operating with class data files copy the data from the file corresponding to next day and include it after the end of the class file corresponding to the current day.

With these two principles mending incomplete and unmanageable data, we can focus on the base of the project, making the next important factor:

We may need to test more than one approach.

This will mean more than one type of modification for the data. Also, modifications may include more than one value at the end of instances, apart from the corresponding class, and even some of the values of an instance may sometimes not be included. We should of course consider the initial case in which we write the corresponding class to each instance, as a modification itself.

-Variables must be created to indicate which kind of modifications the user wants to choose.

-Individual methods will implement each of the modifications.

When we finish modifying the files, the next step will be using MOA with those files. In order to do this, we must face the next problem:

MOA will only accept "arff" files to load data.

We will need to adapt plaintext files to *arff* extension. This is a file extension created by the Machine Learning Project at the Department of Computer Science of The University of Waikato, the same university that developed the tool MOA [17].

The differences between the plaintext files and *arff* files are located at the start of the files, not having to modify the format of data instances. This will mean that:

-When writing data instance modifications, the first step before including the new instances will be defining the name of the file, the variables included with their respective domains and the starting line for the data instances in the file.

This kind of files produces yet more problems related to the different approaches:

Each of the variables included and the domain of these variables must be explicitly stated in "arff" files before any data instance.

This translates to the following statement:

-Before any write, the necessary data to properly initialize the "arff" extension of each modification of a data instances file will be saved.

The number of instances of modified files may not always be the same.

For example, to include the vectors of movement of the mark of a seismometer in an axis from one point in time to the next we need to consider the possibility of using fewer instances as means to be more time efficient.

-A variable must be included to audit the number of instances for each modification file before its creation.

If we pretend to use the whole database, we may need a quite large space on disk to save the data, and even bigger if we need to save numerous modifications of it.

The space available for data may not be as big as the whole extension of data.

It was already decided to work with files of around 1GB, making complex operations including more than one day of data instances multiplies the amount of size needed. Split data from a day of instances into hours (as it was discussed before) also doubles the amount of size needed if we save this data. Another important fact for this next decision is that MOA will need to work with each of these split files in order.

-After creating each split data files, MOA will operate the file until all data instances are used, moment at which the file will be deleted. When all split files of a day have been used by MOA, the data instance files will be deleted as well.

Only after testing the whole system, it was found yet another problem within data instance files.

Other seismometers are added to the instances inside part of the files.

The data found from seismometer "PIL" indicates other seismometer appear inside the files, though the data read from this seismometer is mostly empty, probably indicating malfunction of the device.

-The start of data instance files must be read in order to find out which seismometers are included, to eliminate the information from seismometers different from ABL, CO2, COB, LAR and SHA.

After taking into consideration these rules, we can create the code for this part of the system, which we will include into its own class called *DataOperator*. There are of course multiple designs that can arise from the rules given. The one selected will be shown in the section 5.7 (Implementation).

5.5.2. MOA operator

With this class object it is intended to use modified data files as input stream for algorithms included inside MOA [16]. We will also measure the time each algorithm of the group selected takes to learn from instances and, from their predictions, compute the confusion matrix depending on the type of modification of data.

Let's start then by setting streams for MOA and analyze problems derived from this.

We may not know which file we will include as input for the stream at the beginning.

We will need to set one file for the stream before choosing the right one, although we will be able to change as many times as needed the file of the stream.

-A mock file will be set as initial stream before changing it to any of the modified instance data files.

This time the problem was solved by including a file with one data instance called *example.arff* working as a placeholder.

If we intend to use the algorithms within MOA we must comply with the following:

Stream objects and algorithms need to be initialized before their use.

Algorithms need to set the stream object before any operations over data.

Basically, we need to initialize and relate streams with algorithms. We will combine the solution to problems related to streams into one method and the ones related to algorithms into another.

-When setting new stream files, a method will be necessary to initialize the stream and immediately later prepare it for its use.

-When setting algorithms, a method will be necessary to initialize the algorithms, prepare them for it use and select the stream of each one (which can be shared).

In MOA, streams and algorithms include a method called *prepareForUse* which denies the possibility of use if not called to initialize fields within these objects. It will be needed for these methods.

Each of the files related to the results or structure of an algorithm will require a name.

We will include create a list of filenames related to algorithms and save space by setting the names inside last method.

-A list of filenames for the results derived from algorithms will be created and initialized before any operation over data.

The results of the algorithms must be kept with their own structures as well:

Classes of data instances for each modification type must be differentiated and set before any operation over data.

Depending on what we may try to prove, the domains of classes to which each data instance belongs can be quite different between them.

-A list of classes for each modification will be created and initialized before any operation over data.

To compute the proportions of data instances by class, the number of instances of each class and instances predicted correctly per class must be kept before any operation over data.

This will be useful for indicators of rate of accuracy for each class and crucial to derive cases in which sampling adjustment may be necessary.

-A list of samples per class type will be created and initialized before any operation over data.

-A list of correctly predicted samples per class type for every algorithm used will be created and initialized before any operation over data.

Predictions over data instances must be audited for each of the algorithms used.

This can be translated to the structure of a confusion matrix.

-A confusion matrix of correctly and incorrectly predicted instance classes will be created and initialized before any operation over data. Each algorithm used will have an associated confusion matrix for its results.

We may need to differentiate the speed at which algorithms learn and test and compare them.

In order to prove that a reliable algorithm is able to run fast enough to create a real-time application for the prevention of seismic events, we should also compute the time taken to process the data.

-For each modified file used for training or testing, the average value of the time taken by each algorithm to process an instance must be saved.

In section 6 (Experimentation) the reader may have a better understanding of the reasons behind some of the guidelines included, such as next one:

Undersampling for "NULL" class instances may be necessary.

-With the use of the list of correct number of samples per algorithm, a method to indicate when an instance must be used for training to equilibrate the proportion of instances of the "NULL" class must be included.

The visualization of results needs to be upgraded to be able to deploy graphics.

Previously it was mentioned the usage of *csv* files to upgrade the visibility of data. Inside this class we will include the methods to transform the data.

-Rates of accuracy, confusion matrixes and average learning & testing times per file must be computed and saved in a *csv* file for each algorithm used. The separator used for the values inside files will be a semicolon.

To prevent errors inside these results, there is an important aspect from the modified files that is easy to forget:

The number of instances of modified files may not always be the same.

-Before operating each file, a variable must contain the number of instances to be read.

The next of the predicaments comes loaded with a list of design requirements. It may be simple to understand but its implementation requires multiple methods to solve:

Experiments must be able to be stopped and resumed at <u>some</u> points in time.

Note the emphasis on the word "some" indicates we won't always be able at any point in time to stop an experiment. For simplicity and lack of necessity for a more sophisticated experiment stopping functionality, we will allow this just after using the whole contents of a file. Still, we will only be using this function after the analysis of each day of data. This decision may seem random, but the reader may understand better its significance after studying the implementation of the GUI and its relation to the system in later sections.

We will split this problem into multiple subtasks:

The objects containing the algorithms should be saved as separate files.

The algorithms used by MOA can be saved with *moa* format. We will need to take the java object containing each algorithm and write them in their own file.

-To save the state of algorithms, the virtual objects containing them will be written into their own files as MOA *Classifier* objects using *moa* file format.

> *The objects containing the algorithms can be loaded before starting or resuming an experiment.*

Of course, to restart an experiment or using a trained algorithm, loading its contents is vital.

-A method must be included to load into MOA *Classifier* objects the algorithms contained inside *moa* format files. The loading operation must be called only before any call to testing or training methods.

> *The readable models of the algorithms should be saved as separate files.*

While not being obligatory, MOA offers methods to save the models of algorithms in a way that allows programmers to create the same logical structures using other programing languages.

-MOA's method *getModel* must be used to obtain the models of algorithms, which must also be written into their own files, different than the files containing their states.

> *The statistics related to current rates of accuracy, confusion matrixes and number of instances used must be saved inside their own file.*

In this case, a plaintext file containing this information was created, and the origin of the information was this class only. Since its contents aren't usually bigger than the hundred Kilobytes, we will save its data in plaintext with values split by a separator character and objects split by line breaks.

This class will only transform these statistics into an array of values, which will be later written inside a file by another class.

-The statistics of current experiment state must be transformed into an array with the following format, separating each value by a vertical bar character:

a. The first line will contain the total number of instances read.
b. The second line will contain the number of instances learnt by each algorithm.
c. The third line will contain the number of instances correct per class for each algorithm.
d. The fourth line will contain the number of instances per class.
e. The fifth line will contain the total number of samples correctly predicted for each algorithm.
f. Next lines will contain the confusion matrix of each algorithm, using one line for each.

By fixing next subtask we will have finished with not only the problems related to the objective of loading and saving experiments, but all methods necessary for this class:

> *The statistics related to current rates of accuracy, confusion matrixes and number of instances used should be loaded before starting or resuming an experiment.*

-A method must be created to substitute the statistics of any experiment state from an array with the same structure as the one used by the previous method.

5.5.3. File operator

This part of the application depends on the structure of the data base and its management. The reader should understand that this design must then adapt to the data base, making this a very restrictive tool, only useful for this case. However, the problems found may be common and help for a better understanding of data mining systems.

The objectives for this class are simple: making petitions to the database containing the data, correctly downloading the files and saving these files into a folder. In order to solve the first one an analysis of the behavior of the database and its structure had to be performed, by using the web browser Mozilla Firefox, which will be explained inside section 5.7 (Implementation). However, problems found in its developing are included here, starting with:

> *The petitions necessary for downloading files through the website accessing the database are unknown.*

It would be inefficient to download the whole database before starting the application. In some cases we may not have available that many gigabytes of space in our machines, so making it a process of petition-download-use-deletion will be the best solution.

-An initial analysis of the packets send through the network as download petitions of files must be performed in order to understand and differentiate operations for the correct download of each individual file.

> *To use the instance data from 31st of January we need class data from the same day and the 1st of February.*

In the section 5.2 (Analysis of data) it was explained that files required for the application. A problem discovered in the design of *data operator* involving class data files made necessary merging the contents of two class files for each instance data file modification. In this case, we unexpectedly need the class file corresponding to the 1st of February. Luckily for us, while data instance files from that month aren't available, all class files of February exist in their own folder in the database. This translates into a change in design:

-Class data files from 01/07/2015 to 01/02/2016 (including this day) must be downloadable by a method.

> *The server may not always be available.*

If that's the case, we won't be able to keep operating data until it comes back. There are ways to prevent the application to stop and wait for the server.

-If requests to the server don't give the expected results, the same requests must be resent after a determined amount of time.

As an example, let's choose that amount to be 60 seconds.

The format of the name of the files always uses three digits

This may seem easily solvable, and indeed it is, but that doesn't make it a small problem. For example, if we intended to download files from any day of January, part of the query would be translated to the value "1" instead of "001", making impossible those kinds of requests to files that simply don't exist in the data base.

-A method must be able to translate the numeric day of the year with domain values comprised from "1" to "365" (including this value) to a three digit number.

Each individual file has its own URL value for requests.

After the analysis for the first of predicaments for this class, it was discovered that each file had an associated value. The problem lies on how to translate the name of the file we desire to the URL value. This will be explained further in the section 5.7 (Implementation).

-A method must be created to translate class data file names into the URL corresponding to those files.

-A method must be created to translate instance data file names into the URL corresponding to those files.

Before authorization to the use of the data base, the information of an HTTP cookie is required before requests.

It could be compared to the process of getting a key to open the door to a room with all files, with the room being the data base and the key being the piece of information called "HTTP cookie". For a web browser this process is automatically done, but for us we have to do it from scratch. It also gives us another handicap:

The HTTP cookie required for authorization changes after an unknown period of time.

If we used the wrong cookie we wouldn't be able to access the data base until we inserted the right value.

-Before each request to the server, the current value of the HTTP cookie needed for authorization must be retrieved.

A last appreciation must be made about this problem. Imagine that just while we are sending the request the value of the cookie changes. What could we do in this case? Since it was decided before to resend requests if the result is unexpected, this is already covered by the design.

5.5.4. Data mining system

With the initial three classes we are able to perform the whole process of downloading files, modifying files and feeding the data to algorithms for training and testing. This would require constant monitoring to select and start operations from every class, making it slow and tedious. To fix such headache, another class was created to give the orders to every component and control outputs such as information printed and errors sent by them.

Printed data about the behavior of the application may be lost after execution.

Supposing we included this printed information (which may be optional, but quite useful), we may want to produce and save this data to understand the order of operations taking place, files used or number of requests sent to the server.

-All printed data from console output in a session must be saved into its own plaintext file.

Information about errors occurred in the application may be lost after execution.

Java programming language uses another console to output the errors, preventing printed information to merge with this data if the programmer desires so.

-All error data in a session must be saved into their own plaintext file.

When starting a new experiment after a previous one has finished, the files produced before can be overwritten with data unrelated to the last experiment.

Letting the user handle this problem would be an option, but there are other ways to allow these files to coexist with files from other experiments.

-When an experiment is finished, all related files produced except for the data printed by console or related to errors must be renamed, adding the date at which the experiment ended to the current name of files.

The three classes contained must be initialized before their use.

Methods for the initialization of these components were already covered by the design, but that doesn't mean they are explicitly called!

-A method must be created to contain the calls to initialize the classes contained before any operation requiring their use.

The rest of the methods for this component are those strictly related with the order at which operations take place and what the user may need from the application. Nevertheless, this part of the application won't be excluded from the document, but instead described in more detail than design may convey in the section 5.7 (Implementation).

5.5.5. Purace GUI

The author ends the section with the most abstract component in terms of results from one application to other: the graphical user interface. Its functions are to give users unrelated to the programming of the application the possibility of generating experiments, and also create a way to produce new experiments faster and more interactively than directly programming the behavior required by users for each session.

To aid us drawing the interface in the screen we will use the Java library "Swing". There are of course other alternatives, but since this library was used in prior projects it will be faster to implement than others in this case. Inconveniently enough, a portability problem was found after producing a version of the application.

Swing library behaves differently depending on the operative system.

Things as image positions and font size vary, for example, from Windows platforms to GNU/Linux platforms. In this case the code was adapted to Windows and later a version was made for the training and testing of time consuming experiments for GNU/Linux. For future applications it's recommended checking the Operative System to set correctly the variables from this class.

-The interface must be tested with the Operative System selected for the use of the application to prevent misplaced images and content.

Each experiment can use a different modification.

With something as simple as buttons in the window displayed we can accomplish this. We will also include which objects and behaviors we expect for this part of the interface.

-The interface must contain buttons to select the modification used by files in each experiment.

-A field in the window displayed must contain the current value selected of the modification type.

Days selected for experimentation and whether they will be used for training or testing or have already been used must be shown inside the window displayed.

Deciding on what type of structure will display this information depends on the interests of the programmer and the end users. Some structures can define their function implicitly better than others, but sometimes doing so can make the interface tedious and complicated. Let's see an option:

-Each day used for the experiment will be defined as a square button containing a numeric value to define states unused, training, test and used. These buttons which will change its color depending on the value inside the numeric variable, and this numeric variable will rotate between values of their domain with each use (or click) of the button.

If we decided to just show last buttons as an array, another problem could occur:

Buttons dedicated to the state of days of the experiment are not differentiable depending on the month or the day of the week they represent.

-Buttons dedicated to the state of days of the experiment must be organized as a calendar, changing its position depending on the day of the week and the month they represent.

There is a way to comply with the design that would generate even another problem:

Is not possible to know which month buttons dedicated to the state of days of the experiment represent, even if it's possible to differentiate which belong to one month and which belong to the next.

-A field must be included to describe the month represented by grouped buttons dedicated to the state of days of the experiment.

Selecting weeks or whole months may take too much time by clicking the buttons representing each of the days.

Imagine how much time would take any user to choose the value of the 184 days by clicking each of the buttons!

-By pressing a keyboard button while clicking the button representing a day of the experiment, the numeric value of that day and the variables representing next six days must be changed. By pressing a different keyboard button while clicking the button representing a day of the experiment, the numeric value of that day and the variables representing the days from the same month must be changed.

In this case, for the first key the shift key was chosen, and for the second the control key.

The user may not know the meaning behind the colors selected for the state of buttons dedicated to the state of days of the experiment.

-When starting the application, an image will describe the colors of the different states of buttons dedicated to the state of days of the experiment.

The user may need an example experiment setting to understand how the interface works.

-Buttons to initialize values of the interface as to display preset experiments must be included with a field briefly explaining the option (or the experiment setting) we can choose by pressing it.

It is time to remember two of the specifications from *MOA operator* referring to loading and saving the state of an experiment and add functions to upgrade that aspect of this application.

Experiments selected should be loadable from and savable into a plaintext file.

-A method must save the state of the experiment along with the state given by the class MOA operator into a plaintext file. This state will be written using the first line of the file with the following format:

a. The first character will contain the number of the modification selected for the experiment, followed by the character "|" as separator.
b. Next, the numerical value of each of the buttons describing the state of each day possible for the experiment will be included in ascending order by date, separating each value with the character "|".

-The state of the experiment saved into a plaintext file will contain from the second line onward the data given by MOA operator with the same format as it is obtained.

-A method must read the data of the save state generated in the plaintext file and initialize the modification field and the state of buttons representing each day of data possible for the experiment. Next it will send to the *MOA operator* class inside the *data mining system* class the data necessary to load the state of the experiment as the array read from second line onward from the plaintext file.

-A button must be created in the display to load the current saved state of the experiment. If pressed, the loading operation will be executed before resuming the operations required for the experiment.

We need a mechanism to start and stop experiments.

-A button will be created to start the current selected experiment in the event of being pressed the first time.

Why only the first time? The reader may ask. The answer lies within the solutions next problem. But now let's focus on the task at hand…

-A button will be created to stop the current running experiment after the process corresponding to the current day selected has been finalized. The state of the experiment will then be saved.

We need a mechanism to resume stopped *experiments.*

-When the starting button is pressed again after the experiment is stopped, the experiment will continue from its current point.

-When the starting button is pressed again while the experiment its finishing the operation related to current day and the stop button has been pressed (indicating imminent stop) the experiment will negate the order given by the stop button and continue with the experiment as if that order wasn't sent.

There isn't any way of knowing at first glance if the application hasn't started, is running, has stopped or has finished the current experiment.

-An image must display the current state of the application at all times depending on the orders given by the buttons corresponding to “stop”, “load” and “start”.

When operating data, there isn’t any way to know the progress of the operations related to current day.

-An array of 24 squares must indicate the number of currently used and unused files created by the modifications of the original instance file by the *MOA operator* class. The array will be displayed horizontally, and squares will be initially painted blue, and painted green from left to right as the *data operator* class starts creating a new modification file, except for the first square. For each file which related processes have started, the next of the squares will change to green color. When the 24 files have been used and as a new file containing a day of instance data is requested to database, the squares will change to their original state and start coloring green the first square as the operations related to the file from the next day selected for the experiment begin, until the second of the modification files starts being generated.

If the updates to the interface and the process of data mining are done by the same thread, the interface will lack interactivity.

As it is usually done for interfaces, we will split the work between various threads.

-The data mining process and the processes related to the interface and control of the application must be inserted in different threads.

Although the design states at least two threads and there are ways to create the same functionality as the one from this real application with only those threads, the case included is slightly different from this option. It was decided to use one thread for the control of *data mining system* class operations, one for changes produced in the interface by the user and control of the application and another one for changes produced in the interface by the changes in the state of *data mining system* class operations.

The only parts left to include would be the methods to update the interface and the method used to launch and select the experiments. Since the factors that define those functions also depend on the needs of users and programmers, it will be left to them this part of design. Still, that doesn’t mean an actual example of these parts of the application won’t be described, but that will have to wait for the section 5.7 (Implementation).

5.6. ARCHITECTURE

In this section it will be explained the relations of each of the classes created within the system. As the reader may have guessed from other sections, the core of the program is structured by layers which are the classes *FileOperator*, *DataOperator* & *MOAOperator,* which belong as part of the class *DataMiningSystem*. The flow of operations inside this class is divided into the following steps for each of the data instance files containing one day of data:

- Step 1: the *File Operator* makes petitions to the database to retrieve files which structure is defined as indicated inside section 5.2 (Analysis of Data). This gives the necessary objects required by the *Data Operator* to start its routine.
- Next, for each hour in a day, we perform steps 2, 3 and 4:
  - Step 2: *Data Operator* reads the original files and creates a new file by operating the data with a chosen group of modifications, giving the necessary objects to *MOA Operator* to start its routine.
    The programmer can choose which of the used files to delete at any point in time.
  - Step 3: *MOA Operator* uses the created file to feed instances to different Data Mining algorithms chosen by the programmer, and creates (or in case they exist, modifies) files with the results given by the algorithms and saves their structure.
    The programmer can choose to retrieve statistics as a way of saving the state of the experiment at any point in time. In this case it is done always as step 6.
  - Step 4: *Data Operator* deletes the file created at step 2. Only if we have reached the last hour of the day we proceed to step 5 instead of step 2.
- Step 5: *Data Operator* erases the three files retrieved by *File Operator* which were used in step 2.
- Step 6: If allowed by the user, *MOA Operator* creates a save state for the experiment, which with the files from step 3 give the possibility of stopping an experiment to continue with it at a later point in time. This at the same time makes closing a session or powering off the computer between parts of the experiment conceivable.
  The cycle starts again with next day selected for the experiment if there are more days chosen for it and the user allows continuing the experiment.

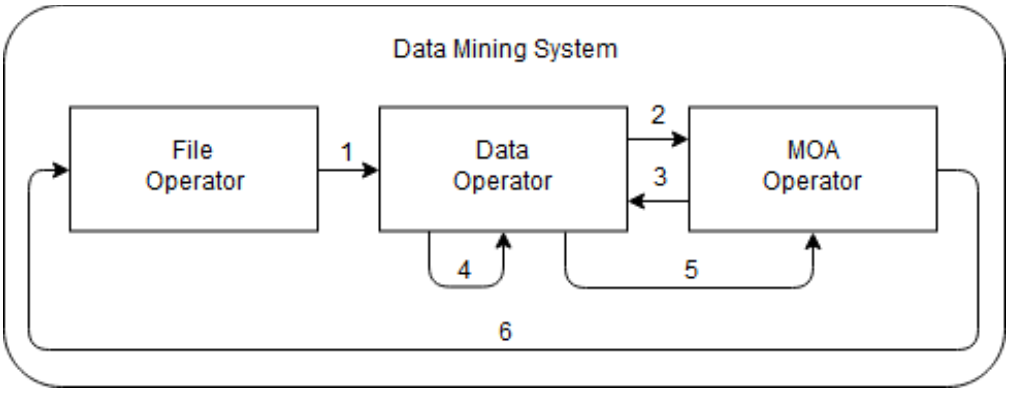

Figure 9: Flow of Data Mining System for Purace application

The function of the Data Mining System class is to initialize and send orders to the core classes, creating the flow of operations indicated before. With these classes we have a working system for the original proposal, but to create, launch and stop experiments we have to program each of them individually. How can we speed up this process? This is where the Graphical User Interface comes up.

As stated in the design, we will divide the interface in three threads:

- The first, controlling the Data Mining System class.
- The second, displaying the window for the interface, applying changes depending on user input and sending orders to the class from first thread.
- The third, updating the window with changes produced by the class from first thread and sending consults to this class.

We also need structures for shared data between second and third threads to comply with the design, depending on the necessities of the user. With their inclusion, we are now able to build the whole system.

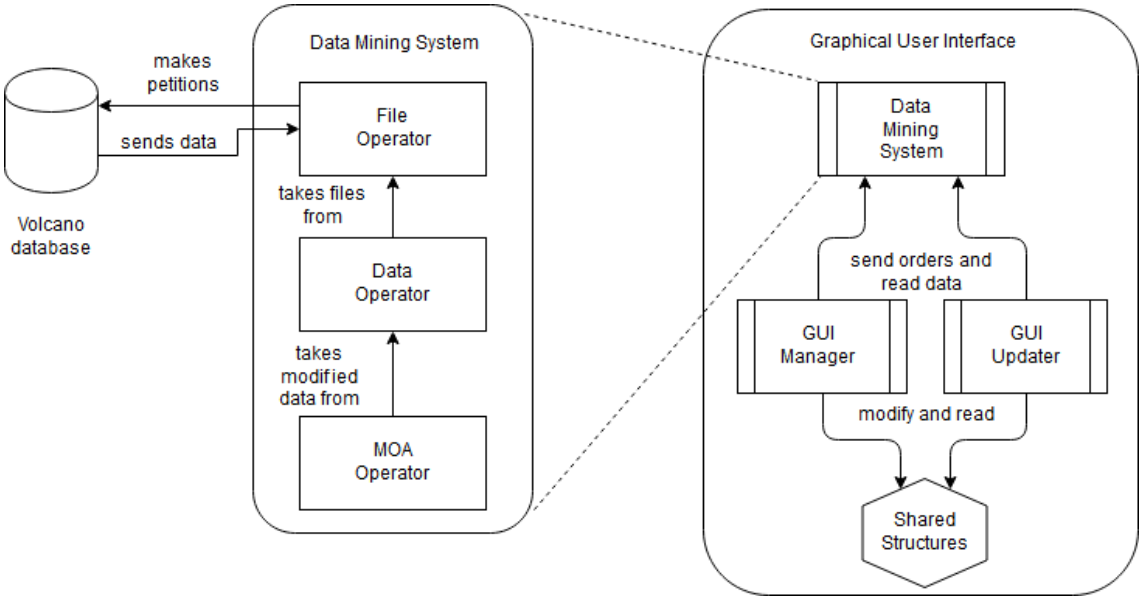

Figure 10: Flow of Purace application's system

This architecture allows debugging more easily any errors produced and also encapsulates modifications of each of the components without inquiring in unexpected changes in other components, as long as the design is followed and shared variables and methods maintain the same interface calling structure (referring to the naming convention for these methods).

The reader may want to understand how a real application is built using the wide list of indications described, and while explaining character by character the contents of the application may be quite time expensive. However, the core methods that make it work will be described to satiate the curiosity of others. And so, section 5.7 will take care of it.

5.7. IMPLEMENTATION

This section covers the essentials of the implementation, giving the necessary clues to program from scratch the core parts of the system. The reader may wonder why explaining only part of the code. There are two reasons for this: lacking time to explain the thousands lines of code (since it's more efficient showing only core functions) and offering the possibility to, through abstraction, create better systems from the ideas presented here to solve not only this problem, but others requiring Data Mining techniques.

We will go class by class in the same order of appearance as in section 5.5 (Design):

5.7.1. Data operator

Using the design included, many kinds of implementations can be constructed. In this case the author will show part of his own; the pillar method of this subsystem. With this we will cover the most important aspects necessary to modify data files.

Let's now explain what each of the lines of this method computes.

```
1  getHourData(hour){
2      if (!variableNamesSet){
3          setVariableNames();
4      }
5      hourOfTheData = hour;
6      prepareData();
7      prepareClassFiles();
8      if (!openDataFile(fileToOpen)) {
9          System.exit(-1);
10     }
11     setModifications();
12     writeData();
13 }
14
```

Figure 11: Pseudocode for method GetHourData from the class DataOperator inside Purace application

The method *getHourData* receives the value of the hour of the day we are going to modify, which can be between zero and twenty-three. The day we have picked is already selected before calling this method, and the necessary files have already been provided by the *file operator*.

We talked about how we need some necessary values at the start of an *arff* file depending on the modification. From lines 2 to 4, in case these values are not initialized, we set them inside the method *setVariableNames*. We also set variables containing the number of variables inside each kind of modification of the data, since it will be used for computing part of the logic of the program.

In line 5 we save inside the variable *hourOfTheData* the current hour we are going to modify, which is given by *hour* variable at the beginning. This variable is global, so it will be read later by some of the next methods in this process.

In line 6 we initialize variables including the names of the files we are going to modify and create with method *prepareData*. It's important to note that here, we will also choose the hour of the day we will read from the data instance file.

In line 7 we merge class data files of the current day selected and the next one, but only in case we haven't done it before with the method *prepareClassFiles*. We also save each of the instances of the merged class file into an array for later use.

Inside the method *openDataFile* in line 8 we read the values of each instance corresponding to the hour of the day selected and save this data into an array of instances. Each of these instances contains an array of values within itself which won't be complete after this method executes, since some of the values must be created as a modification of the data instances later. We must also deal with the last problem included in the design, eliminating depending on the file, parts of the information recovered. Depending on the modification scheme chosen, instances will also have a different structure.

From lines 8 to 10 we check if the operations with the data instance file were successful, and exit from the system in case they weren't.

The method *setModifications* from line 11 includes the values from instances not yet computed depending on the kind of modification we want to apply. This method checks each of the instances one by one and with the data given by the merged class file decides the class we need to assign to each instance, and also other values depending on the decisions taken.

In line 12, by using method *writeData* we create the *arff* file which will contain all modified instances equivalent to an hour of data.

With this function completed, operating a whole day of data is trivial if we have the files; we just need to call the method one time for each hour in a day, of course setting before the value of variable "hour" to the one of the hour we desire.

Methods related to deletion of files haven't been explained because of their simplicity, but the author reminds they must be included to comply with the design.

5.7.2. MOA operator

Only one method of this class will be included as pseudocode, which is actually the most important: the method that feeds modified data of one hour of measures into algorithms in order to make them learn from the data and create its own internal structure for predictions.

After understanding its behavior, we will see an easy way to transform it in order to use the same structure of operations for a method valid for testing algorithms.

```
1  learnHour(day, hour, mod){
2      prepare(mod);
3      setHourStream(day,hour,mod);
4      setCorrectNumberOfInstances(mod,hour);
5
6      while (numberSamples < numInstances){
7          trainInst = stream.nextInstance();
8
9          if (!undersamplingNull(trainInst, mod)){
10
11             for(algorithm = 0; algorithm< arrayOfAlgorithms.length; algorithm++){
12                 correct = arrayOfAlgorithms[algorithm].correctlyClassifies(trainInst.instance);
13
14                 arrayOfAlgorithms[algorithm].trainOnInstance(trainInst);
15
16                 if (correct){
17                     numberSamplesCorrect[algorithm]++;
18                     addToSamplesCorrectPerType(trainInst,algorithm,mod);
19                 }
20                 addToConfusionMatrix(trainInst,algorithm,mod,correct);
21
22             }
23         }
24         addToTotalSamples(trainInst,mod);
25         numberSamples++;
26     }
27 }
28
```

Figure 12: Pseudocode for method "learnHour" from MOAOperator class inside Purace application

The method *learnHour* initially takes the value of the day and the hour to which the measures from the file we are going to use correspond (using variables *day* and *hour)*, and the modification type of the data (using the variable *mod*).

In case the variables we are going to use aren't ready, the method *prepare* sets this data. We need to include inside the call the type of the modification as a value, since this class may need to use different data and structures depending on how each data modification structure is conceived. Right after this step we set the file of the stream for algorithms to the one we are going to use specifying again the values of *day*, *hour* and *mod*.

Next step is directly affected by the modification type and the hour chosen. For the cases discussed in the section 5.3 (Modification types), in which creating instances requires data from two different days, we decided to discard its use. This means that some files include a different number of instances than expected, and this number must be included into our variables to avoid errors before using those instances. This is the reason for using *setCorrectNumberOfInstances*.

The following operations are performed until the number of samples used is greater or equal to the total number of instances that will be used in the experiment after the inclusion of instances from the current file (inside the clause from line number 6).

We set the variable *trainInst* to take the value of the next instance prepared from the stream used.

Next, by using the method *undersamplingNull* and introducing the instance and the modification type chosen, we are able to identify the class of the instance (which domain depends on the modification type) and compare the number of instances used that were and weren't of "NULL" class for modifications 6 and 7 and of "-1" class for modification type 8. In the case the number of "NULL" or "-1" instances are lower than a proportion of the number of

instances not belonging to this class (proportion decided by us depending on our interests), the following operations are produced for every algorithm initialized:

> We save inside variable *correct* whether the algorithm correctly predicts the value of the instance or doesn't. The algorithm is then trained with this instance.
>
> Only if the algorithm predicted it correctly, we increase the value of the corresponding variables to indicate that a sample was predicted correctly and that a sample of the class to which it belongs was also predicted correctly. This is done to compute the total prediction rate and the prediction rates by class.
>
> In the end of this clause we add the necessary values to our variables describing the confusion matrix of the algorithm. These values must be computed for each instance used to train since we must also count wrong predictions.

After this set of operations we add one to the total number of instances inspected and increase the value of the variable *numberSamples* in order to control the limit of instances retrieved from the file. From this point until this limit is met, we will come back to the point at which we save the value of the next instance of the stream and keep onward.

This method is of use for training algorithms, but to test algorithms we need to create another method. But the difference between a testing method and a training one is so simple we won't need another piece of pseudocode. It is as simple as not training the algorithm (deleting line 14) and not calling the *undersamplingNull* function (deleting the clause in line 9). The second of this changes means that we will always analyze the instances without mattering the number of instances by class we have used.

5.7.3. File operator

To correctly manage initial files, the first two guidelines of design for this class must be well understood. Applying the first one incorporates the "how" to the program, and applying the second one, the "what". But first, we won't be able to do any request without doing an initial operation specified by the last two guidelines.

Using Firefox navigator developer tools, the events that occurred when different files were downloaded were analyzed. Using the "network" function of this tool, it was discovered that the first message sent when entering the database asked for the value of a variable, and then the server answered with that value. This variable is commonly stated as a *cookie* in the jargon of computer networking, and there can be multiple cookies used within requests for any kind of website. Given this cookie, why would its value be important? The answer lies inside the rest of request produced when accessing and downloading any files from the database; all share a cookie field with the same value given by the answer to the first message. Without this value, no file can be downloaded.

Now, supposing we spend some time using the website to access the database, eventually it will tell us we lack permission to do any operation, and the whole process of accessing the website must be restarted before any new operation is requested. If we analyze again the first message sent after this event happens, the value given for the cookie will change from the one we had before, so any new operations must include the updated value of the cookie.

With this information, let's imagine we want to create a method to download any kind of file from the database. Figure 12 should serve as an example of how to do it:

```
downloadFromURL(desiredFileURL, saveAs){

    try {
        OpenConnectionToWebsite(connection,databaseWebsite);
        ChooseConnectionMessageType(connection, "GET");
        ChooseConnectionTimeout(oneMinute);
        ConnectionRequest(connection);

        cookie = GetFieldFromConnectionRequestAnswer(connection);
        Disconnect(connection);

        OpenConnectionToWebsite(connection,desiredFileURL);
        ChooseConnectionMessageType(connection, "GET");
        ChooseConnectionTimeout(oneMinute);
        AddConnectionProperty(connection, cookie);
        ConnectionRequest(connection);

        ConnectionWriteResponseInFile(connection, saveAs);
        Disconnect(connection);

    } catch (Exceptions) {
        return false;
    }
    return true;
}
```

Figure 13: Pseudocode for method "downloadFromURL" from FileOperator class inside Purace application

The method *downloadFromURL* initially takes the URL of the file we are going to download (*desiredFileURL*) and the name of the file that will save its contents (*saveAs*).

This method will return a boolean value when called. This value will be "false" when an error has occurred and "true" when all operations inside the method are successful. The clause try-catch takes care of turning this value to false as soon as an exception or error occurs within the operations of the method.

We start then by opening the connection "*connection*" to the website of the database. Let's choose the type of the message to send to be "GET" and the timeout to be one minute (which means that it will be considered as an error if the connection is not stablished within one minute of its start). Then, we start the connection to get the value of the cookie that was previously described and disconnect.

If there were no errors in this process, we continue by opening a new connection, this time using the website URL to the file we want to download. Again, the type of the message to send is "GET" and the timeout is one minute. The value of the cookie is added as a property of the connection, and then the connection is requested.

If we receive the expected response, we will save it as a file with the name included in *saveAs* inside the machine used. When this last operation is finished we close the connection and return the value of "true" as proof of completing correctly the procedure.

The value of the variable *saveAs* can be the one we desire, but the value of *desiredFileURL* depends directly on which file we are going to download. To find out the value required for each of the files, the analysis of the website was performed. From it, the structure of its contents was devised. Here a simplified scheme is included:

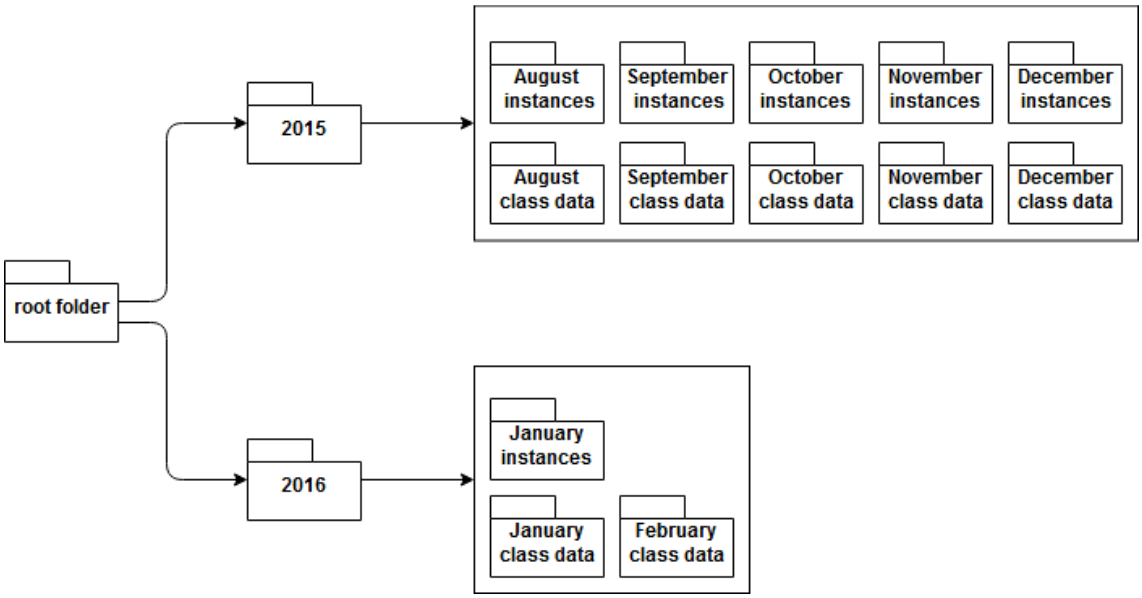

Figure 14: Simplified structure of the database for the measurements of volcano Puracé

The files are divided in folders by the year to which they belong, and inside those folders they are divided by the month to which they belong and the kind of files they contain (instance data files or class data files).

While comparing requests between files inside one folder and another there are very clear differences, doing so within a folder the contrary occurs: all files inside the same folder share most of their URL field with the exception of at most three characters; being the folder any of the reader's choosing. For us this means that knowing the URL field of each folder and finding out how to manipulate the changing characters to select files would give us the complete URL address list for every file.

On the subject of character manipulation, a good start to device a formula for it will be analyzing the differences between files within the same folder, in this case focusing on files related to data from the month of October in the year 2015. As it turns out, if we want to request files related to, for example, day 275 and day 276 and they belong to the same folder, the only difference is within one of the characters, being the one from the first file "5" and the one from the second "6" and both occupying the same position of the field. If we perform the same analysis with files related to, for example, day 275 and day 285, the difference is again within one of the characters, having the first a value of "7" and the second a value of "8", but this time in another position. If we perform a third analysis with files related to days 291 and 301, this time the difference is within two characters, being the values of the first URL "2" and "9", and the values from the second URL "3" and "0", only this time the position of the first character is the same for both files, but the position of the second character those for both files is in the same position as the one in the second analysis.

After other comparisons to prove formulas it was found out (as the reader may as well have) how to fill these changing characters. Is as simple as writing the number of the file they are related to, but divided into three characters. This way, the characters for files related to day 284 become “2”, “8” and “4” in this order. Files related to the first of January would have as those characters the values “0”, “0” and “1” in this order; files related to the second of January would have as those characters the values “0”, “0” and “2” in this order, and so on.

Now, by making a list of URL values common for files within each of the folders and a method to fill the changing characters, we can insert the value of the URL into our method and we will be able to download any file we desire.

To answer which operations are actually called by this class and in what order, let’s look at the following piece of pseudocode:

```
1  getDayData(day){
2      setVariableNames();
3      while (downloadFromURL(getClassFileHTTP(day), saveClassFileA) == false);
4      while (downloadFromURL(getClassFileHTTP(nextDay(day), saveClassFileB)  == false);
5      while (downloadFromURL(getDayFileHTTP(day), saveDayFile)  == false);
6  }
7
```

Figure 15: Pseudocode for method "getDayData" from FileOperator class inside Purace application

The method *getDayData* receives the number which describes the day to which the files we need to download are related. As stated before, two of these files correspond to class files of the day chosen and the next, and the third file corresponds to the data instances file of the chosen day.

We start by setting the necessary variables in case we haven’t already. Then, for downloading each of the files we will use the method *downloadFromURL* one time after another until it gives us back the Boolean value “true”, stating that the operation was completed. When the three files are correctly downloaded the method ends.

5.7.4. Data mining system

The methods described for this class are the ones that call the necessary functions for steps 1 to 5 described in the section 5.6 (Architecture), which strongly advise to review before the reader continues.

The process of training algorithms with the data related to a day of measures of seismometers always follows the same scheme, no matter the modification type or the day chosen. Let’s start with the method that englobes the whole process:

```
1  learningCycleDayMod(day, mod){
2      /*Step 1*/
3      FileOperator.getDayData(day);
4      /*Steps 2, 3 and 4*/
5      for (hour=0;hour<24;hour++){
6          learningCycleDayHourMod(day, hour, mod);
7      }
8      /*Step 5*/
9      DataOperator.deleteDayDataFile(day);
10     DataOperator.deleteClassFile(day);
11     DataOperator.deleteClassFile(nextDay(day));
12     DataOperator.setFusion(notFused);
13 }
14
```

Figure 16: Pseudocode for method "learnCycleDayMod" from DataMiningSystem class inside Purace application

To call the method *learningCycleDayMod* we need to set the value of the day chosen for analysis and the modification type.

By using *FileOperator* we obtain the necessary files for the rest of process, completing the first step.

Next, the helper function *learningCycleDayHourMod* is called 24 times (one for each hour in a day), composed by operations described by steps 2, 3 and 4. When the whole of the modified contents have been produced and used for training, we proceed with the preparations for the next set of files, part of which are included in the next step.

Finally, we call *DataOperator* to delete the three files obtained at the beginning (lines 9, 10 and 11). We also state to this component that the next class data files haven't been fused together (as explained in its implementation) by sending the value indicating this negative inside variable *notFused* as parameter of *setFusion* method.

Let's now focus on the helper function, including the steps 2, 3 and 4 explained in section 5.6 (Architecture), which completes the process of the previous method:

```
1  learningCycleDayHourMod(day, hour, mod){
2      /*Step 2*/
3      DataOperator.setDayOfTheData(day);
4      DataOperator.getHourData(hour);
5      /*Step 3*/
6      MOAOperator.learnHour(day, hour, mod);
7      MOAOperator.saveBothModels(mod);
8      MOAOperator.saveStatisticsAndConfusionMatrixesCSV(mod, day+" "+correctedHour(hour));
9      /*Step 4*/
10     DataOperator.deleteHourCurrentDay(hour);
11 }
12
```

Figure 17: Pseudocode for method "learnCycleDayHourMod" from DataMiningSystem class inside Purace application

To call the function *learningCycleDayHourMod* we need to set the value of the day chosen for analysis, the hour selected and the modification type.

Since we may not know which is the day of measures selected to analyze when calling this function, we need to state to *DataOperator* the value chosen. After doing so, we modify the files retrieved in step 1 and create a modified file containing data instances classified from the day and the hour chosen, finishing step 2. If the reader wonders how DataOperator knows the

modification type, the answer lies within the implementation of the interface, since it's the user who states this value before any operation requiring this function

With the modified file created, *MOAOperator* trains the algorithms chosen by the programmer with the data inside this file. When it has finished, it updates the saved models including the structure of the algorithms and their code (the last in file format ".moa") and saves statistics about the process, finishing step 3. If no models have been produced for the algorithms before, they are created, but if no files related to statistics have been created an error will be produced. This is because it's the user who, indirectly, at the beginning of an experiment, sets the commands that create these files before any operations requiring this method.

Next step is the most simple: deleting the modified file created. We will use *DataOperator* again since this class takes care of deleting all unnecessary files.

As with last class, by changing a few lines we can turn a training process into a testing one, but for that we need to create the testing methods from class *MOAOperator* as explained before, first. If done, by calling inside step 3 the command for testing instead of *learnHour* will produce the method we need for testing.

There is still one step from the architecture that hasn't been explained yet, along with some operations that are set outside of last classes and are still important for the understanding and completion of the application. Following is the implementation of the interface, which contains all the pieces left to explain.

5.7.5. Purace GUI

For a better understanding of the implementation of this class, a few characteristics of the graphics included inside the interface need to be described. Following the design guidelines derived before, the graphical components inside the interface were created, allowing the user to operate the program and create experiments. It must be reminded that there are multiple possible interfaces which could apply these guidelines, making the choice presented an option from many. The following structure will be used for the window displayed when opening the program:

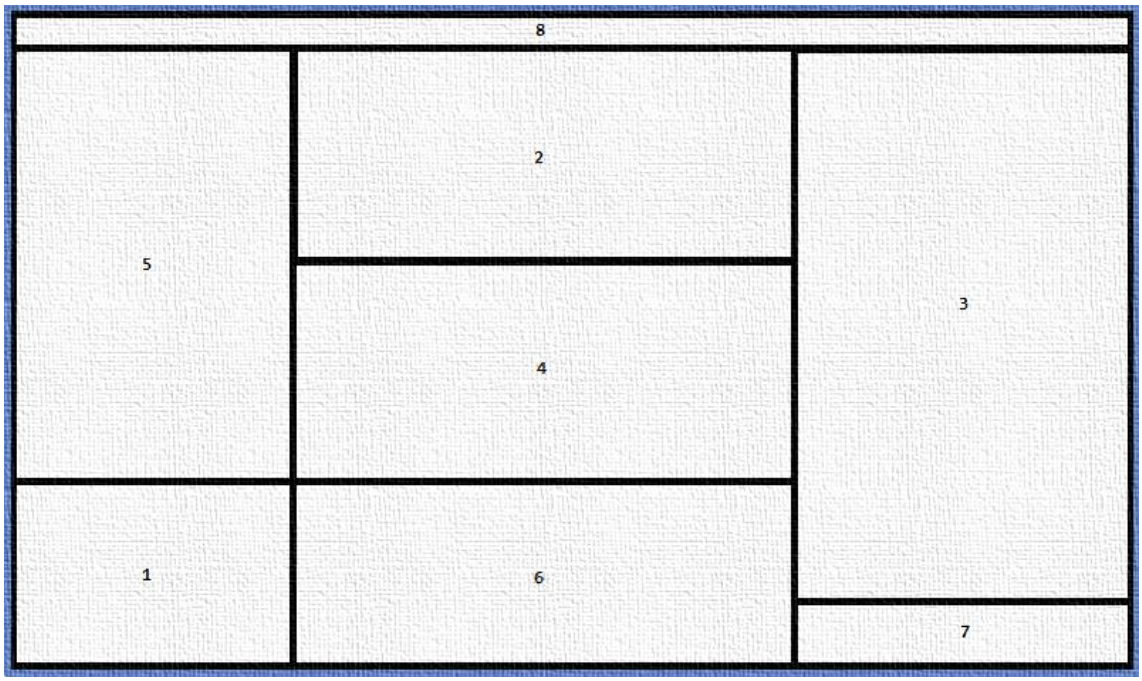

Figure 18: Structure of the interface of Purace application, separated by sections

The interface is divided in 8 sections, each with a different purpose. To explain each of the sections, let's will do it in the same order as in section 5.5 (Design) with the exception of the first and last section:

1. The logo in the bottom-left corner as decoration.
2. The modification section in the upper-middle part. It includes 6 buttons to select the modification of data before the experiment.
3. The calendar section in the upper-right. It divides each of the days that can be selected for the experiments by month. Each of the squares is a button that can change color and value depending on the number of times clicked.
4. The screen section in the middle. It gives information to the user of current events and serves as guide for the domain of values that belongs to buttons from last section.
5. The prepared experiments section on the left, including buttons to set a new predefined experiment using current modification.
6. The operations section in the bottom part with buttons to load, start and stop the experiments.
7. The progress section in the bottom-right corner, with squares indicating the current hour of the day being analyzed in the experiment.
8. The window section at the top, which allows minimizing the window or closing the program.

The result from applying previous design guidelines to the interface structure is the following:

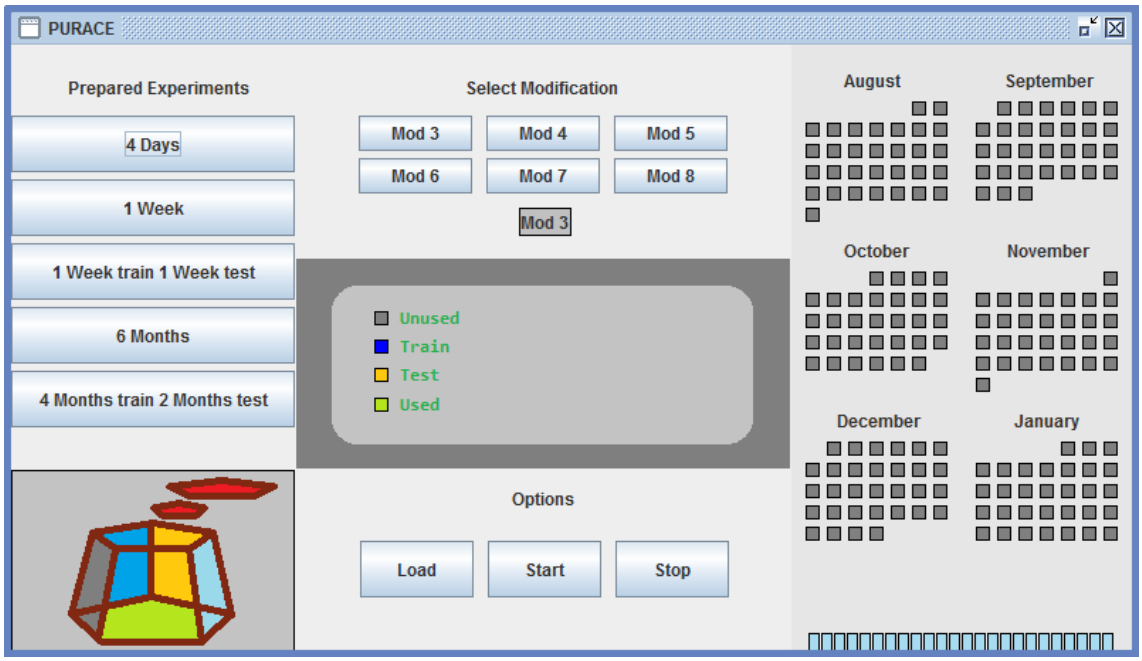

Figure 19: Interface of Purace application

If we intended to create and perform an experiment, there are multiple options depending on our objectives and schedule. We will describe an example of actions taken by any user for creating an experiment, stopping it, reloading it and taking its results.

Inside the folder containing the executable for the application, we will find 3 plaintext files: out.txt, err.txt & saveState.txt. The last of these files will contain the state of last experiment started and will be necessary for its continuation if it was properly stopped before. These 3 files must be included before starting any experiment (even if their contents are empty).

Inside the folder containing the executable for the application there are also 4 folders:

-Data folder, containing the *example.arff* file necessary for the program. Here the data downloaded from the database is stored and then deleted.

-Images folder, containing images used for the interface.

-Models folder, containing the results of finished experiments inside files with the finishing date of the experiment, and the results of partially completed experiments. This folder contains one folder for each of the modification types, using the name "mod_*X*" where *X* is the number of the modification type. These folders serve to split results of using one modification type from results of using another type.

-Modifications folder used as temporal storage for modified files. This folder contains one folder for each of the modification types, using the name "mod_*X*" where *X* is the number of the modification type. These folders serve to split modified data files using one modification type from modified data files using another type.

Without this folder structure the application will crash, so it's important to make sure all folders are in their corresponding location before starting any experiment.

If we start the executable, the window with the interface will be displayed. We can choose using one of the predefined experiments by using the buttons under "Prepared experiments".

Choosing, for example, to click the last of these buttons will produce the following change in the interface:

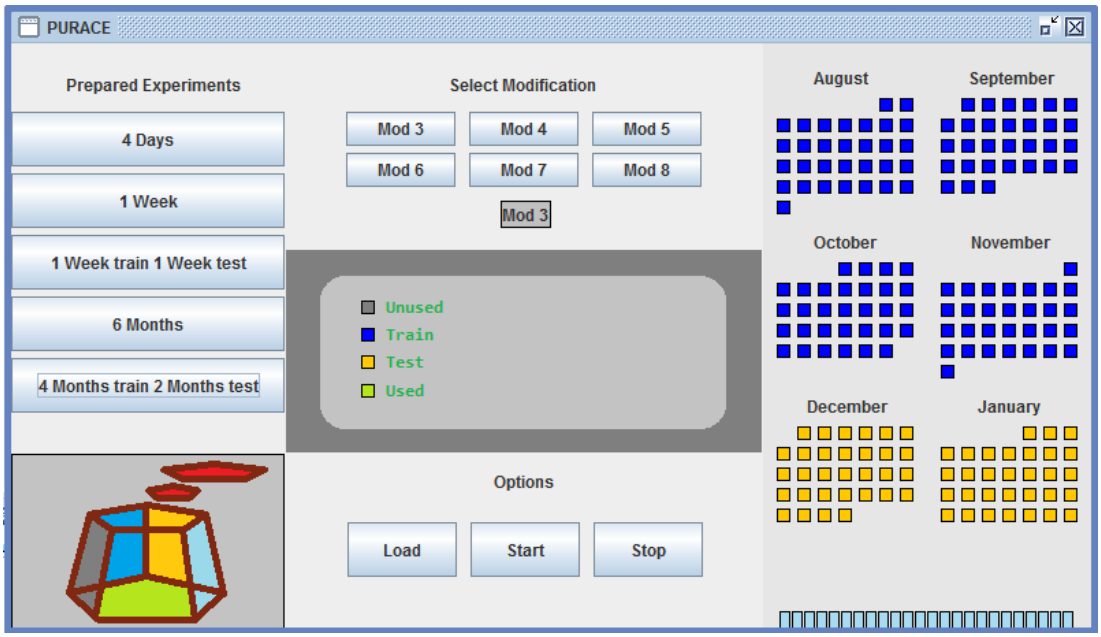

Figure 20: Example of state of the interface of Purace application after pressing "4 Months train 2 Months test" button

Here, each of the squared buttons representing the process for each of the days of measures take a different color and value depending on whether we will be testing or training algorithms using those measures.

We can also choose the modification type for the experiment with the buttons under “Select modification”. Clicking any of the buttons will produce the small dark-grey window under the buttons to adopt the value of the clicked button, to show which modification has been selected. Clicking the button “Mod 6” will produce the following change in the interface:

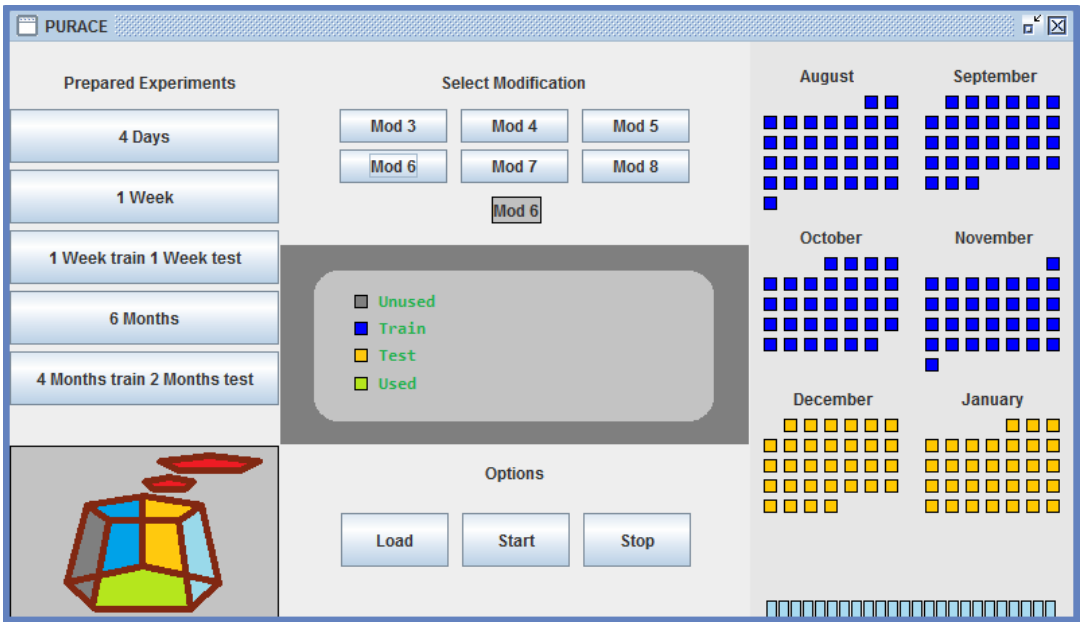

Figure 21: Example of state of the interface of Purace application after pressing buttons "4 Months train 2 Months test" and "Mod 6"

Using the button "Start" under the "Options" menu we are able to start the process of training selected. To differentiate allowed operations from those disallowed for the user, the buttons unusable turn their color darker as the experiment starts (with the exception of the buttons inside the calendar section). As the modified files for the first day are created, used and deleted, the progress section changes to indicate this fact. After the first 14 hours of the first day have been analyzed, the state is the following:

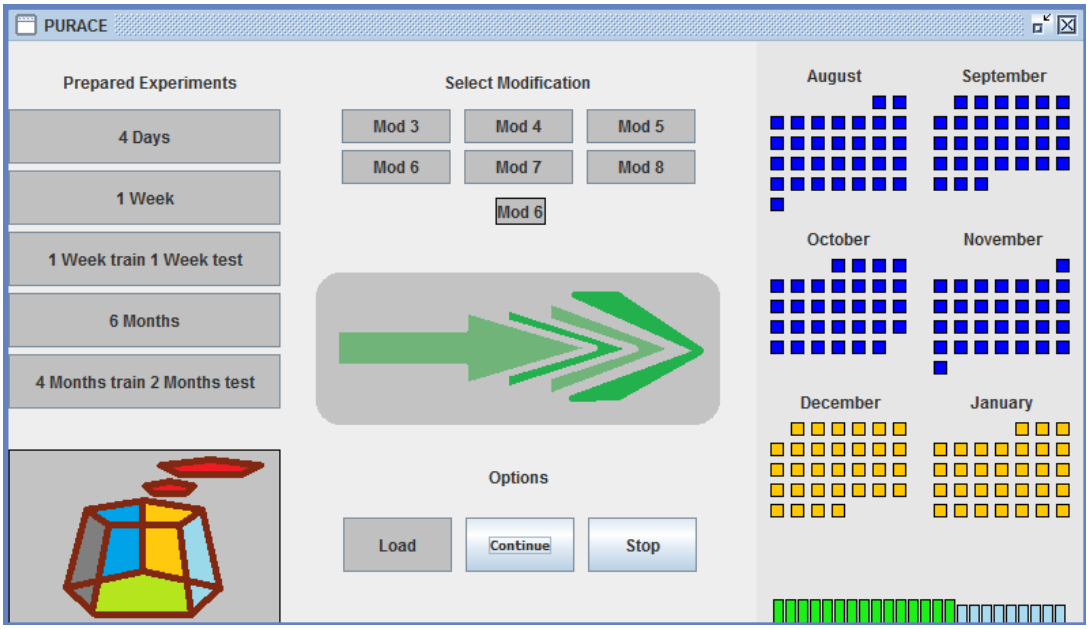

Figure 22: Example of state of the interface of Purace application after the start of an experiment

As stated before, to stop an experiment we need to wait for the current process to finish using the whole contents related to a day of measures. To do this correctly, first we click the "Stop" button in the "Options" menu, which will produce the following change in the interface:

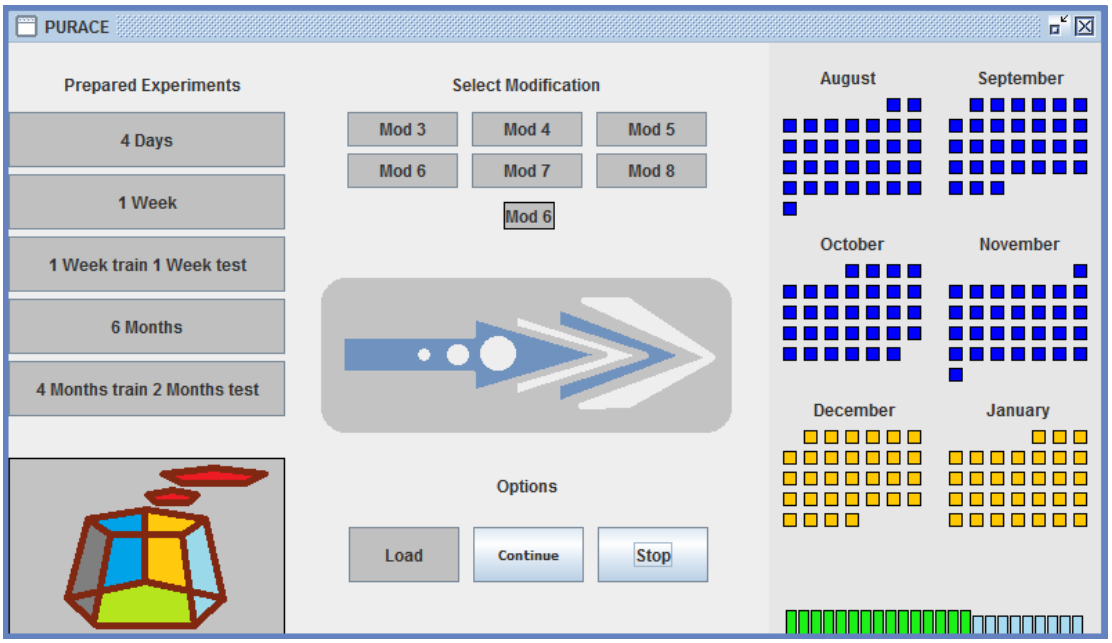

Figure 23: Example of state of the interface of Purace application after pressing the "Stop" button in the "Options" menu

With this change we inform the user that the order of stopping has been read. Then we wait until the operations using the contents related to a day of measures have finished. The user will be informed of this fact with the following change in the interface:

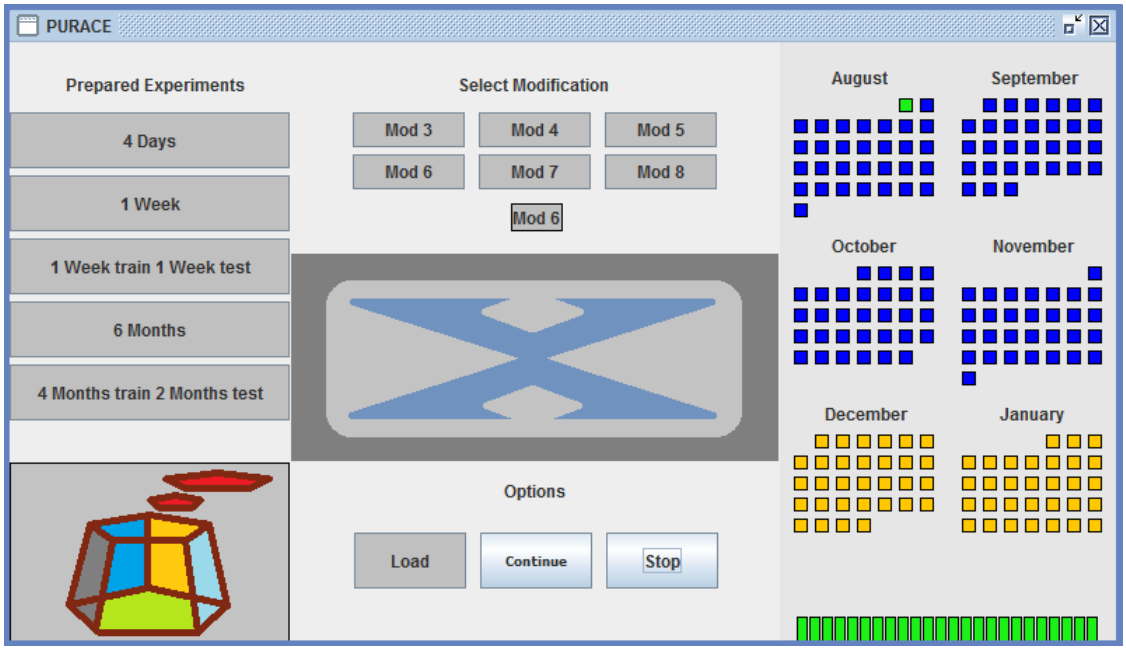

Figure 24: Example of state of the interface of Purace application after an experiment reaches a halt

At this point, we can press the "Continue" button to keep with the experiment. Instead, let's suppose we need to turn off the machine using the application. We would proceed to close the window using the button from the window section to close the program, before turning off the machine. If we want at a later time to continue the experiment, we will execute the program and press the "Load" button from the "Options" menu. To address the fact that the program is prepared to load, the interface changes from the original state to the following:

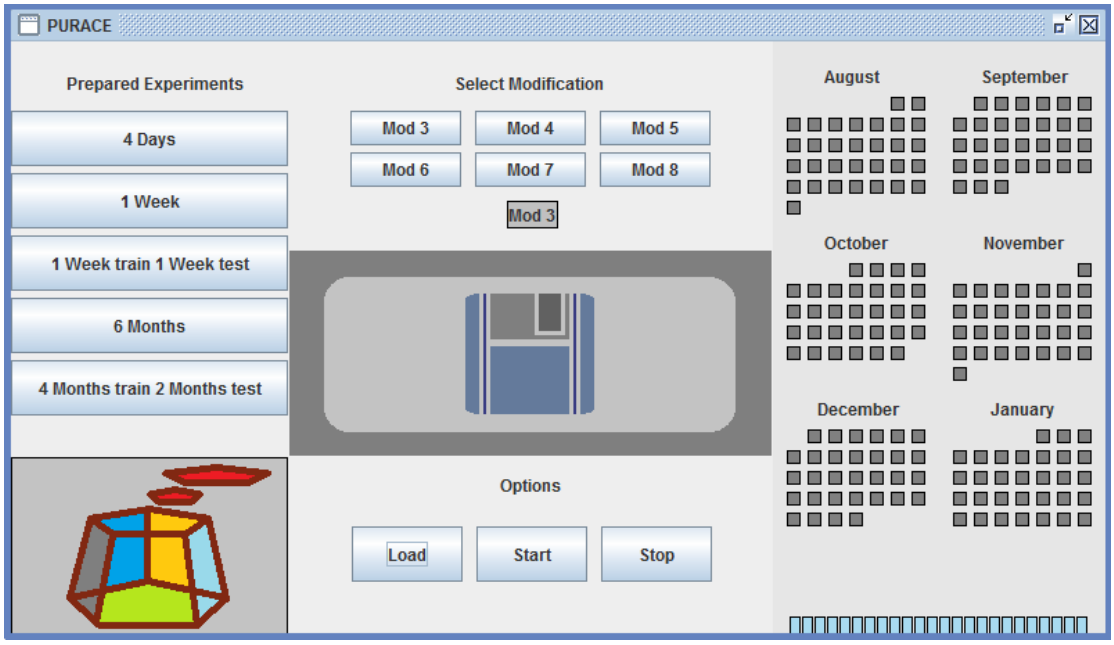

Figure 25: Example of state of the interface of Purace application after pressing the "Load" button in the "Options" menu

Do not mind the current modification selected or the state of the buttons from calendar section, since the loading operation will overwrite these values. If we press now the "Start" button, we will continue with the last experiment partially complete.

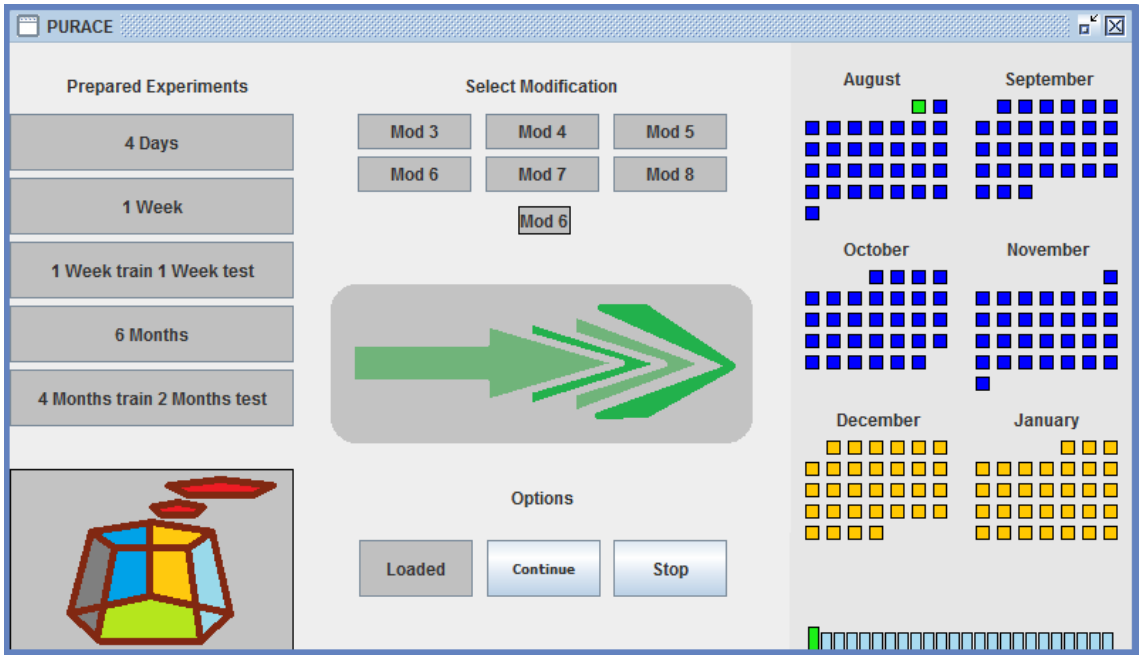

Figure 26: Example of state of the interface of Purace application after resuming an experiment

After the experiment has finished, the interface will have the following state:

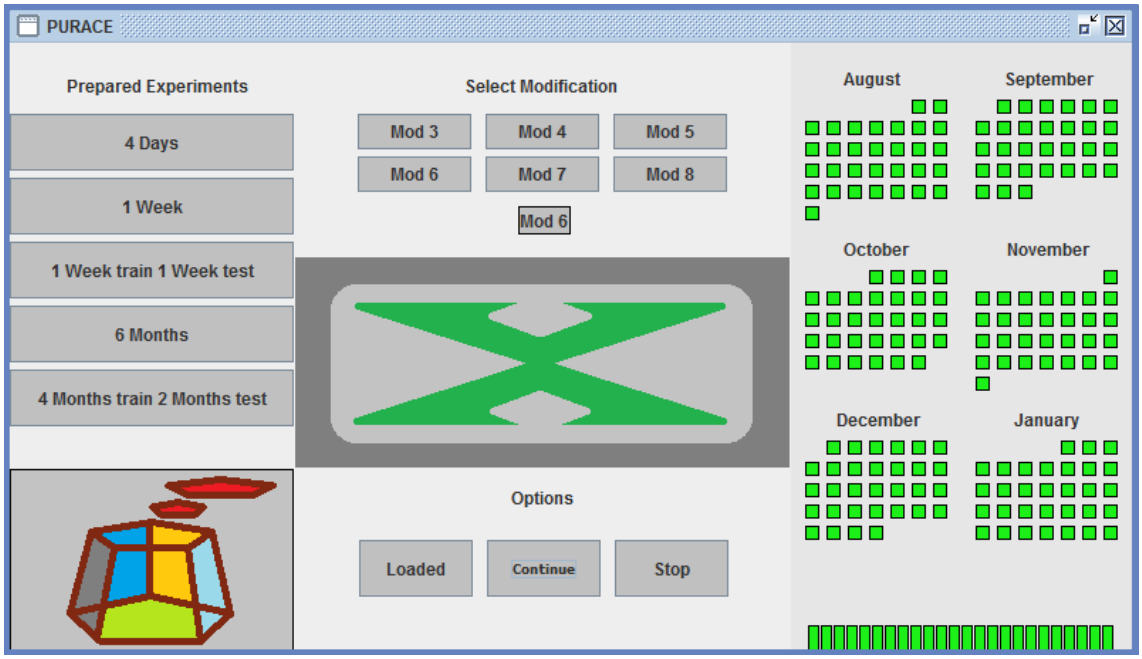

Figure 27: Example of the state of the interface of Purace application after finishing an experiment

We can see that the buttons in the "Options" menu have been darkened, indicating that they can't be used anymore. The only options after the end of an experiment are to minimize or close the window.

There is one last feature included in this application. Imagine at the point at which we were going to load the experiment we wanted to run an old unfinished experiment, which uses the same modification type instead of the current experiment. To load a new partially completed experiment, the files inside the folder mod_6 inside "Models" which doesn't include a date in their name must be substituted by those with the same name from the new partially completed experiment, and the *saveState.txt* from the root folder must be substituted with the save state of the new partially completed experiment. The same can be done depending on the modification type (since there is one folder for each modification type inside "Models" folder), but if the folder for this data is empty, including the files will be enough. Remember also to back up the files being substituted before any new experiment if you want to continue at a later time with the last experiment halted.

After showing the behavior of an example experiment, it will now be explained how orders given by the user transform into controlled operations to create each experiment. We will focus on the core method of this class, which is in charge of managing the states of experiments, sending orders to the system and updating the interface with changes produced by the system while parts of the experiment are completed to inform the user of these facts.

```
1  experimentExample(){
2      setInitialButtonColors();
3      system.prepare(mod);
4      if (!load){
5          system.createStatisticsCSV(mod);
6          noMoreLoading = true;
7      } else {
8          loadState();
9          load = false;
10         noMoreLoading = true;
11         LoadingOptionsButton.setText("Loaded");
12     }
13
14     for (day = 0; day < daysChosen.length; day++){
15         if(stop){
16             saveState();
17             running = false;
18             infoImage.setIcon(stopIcon);
19         }
20         while(stop);
21
22         infoImage.setIcon(runningIcon);
23
24         if (daysChosen[day] == TRAINING){
25             system.learningCycleDayMod(CorrectedDayValue(day), mod);
26             daysChosen[day] = COMPLETED;
27             stateButton[day].setBackground(GREEN);
28         } else if (daysChosen[day] == TESTING){
29             system.testingCycleDayMod(CorrectedDayValue(day), mod);
30             daysChosen[day] = COMPLETED;
31             stateButton[day].setBackground(GREEN);
32         }
33         system.setCurrentHour(firstHourOfTheDay);
34     }
35     running = false;
36     saveState();
37     system.renameExperiment(mod);
38     infoImage.setIcon(finishedIcon);
39     system.closeConsoleAndErrorOutput();
40     changeOptionButtonColorsWhenFinished();
41     finished = true;
42 }
```

Figure 28: Pseudocode for method "experimentExample" from PuraceGUI class inside Purace application

We start by setting the initial state of the square buttons representing each of the files related to the days of measures from seismometers available, as well as the state of other buttons. Then we prepare the system and set its modification type to one of the available.

At this point the Interface will change depending on the actions of the user, but our interest is in how experiments are produced. The only important information to know is that the user has two options before starting the experiment: selecting which files to use for training and testing of algorithms and the modification type of the experiment (with a value by default), or, choosing to load the experiment previously saved.

If the first is chosen, the options will be saved, various statistic files for the experiment results will be created and we will state that no loading will be performed. If the latter is chosen, as soon as the experiment starts, the experiment state will be loaded. In this case, we state to the system that we are not waiting for loading state (line 9) and that no loading operations are necessary anymore (line 10) while informing the user of the completed loading event.

At this point, with the clause from line 14, we perform the following operations for each of the days available to be analyzed which can be chosen by using the square buttons mentioned in the design of this class:

> Initially, we check if the user has given before the other to stop the experiment. If so, we save the state of the experiment with method *saveState* (which would correspond to step 6 included in section 5.6), allowing the user to exit the application when this operation is finished. Then, we state to the system that it has stopped running the experiment and inform the user with an image of this event (named *stopIcon*). The user can then exit the program, but also continue the experiment. In case we continue the experiment, we wait until the user pushes the button which states to the system to resume before doing so.
>
> Either if we have stopped and resumed or just continued with the experiment, we inform the user of the current state of the program by displaying an image (named *runningIcon*).
>
> If the current day wasn't chosen we skip to the next day, going back to the beginning of this clause. If we chose to train or test the algorithms using the data related to the current day, we inform the data mining system to perform the necessary operations while also stating the modification chosen for this analysis. When the analysis is completed we change the state of the experiment and the interface to indicate that the part of the experiment related to the chosen day is finished.
>
> Before skipping to the next day, we state to the system the fact that since we may use a new file next, we will begin any of the next possible analysis by using the first hour measures of the file related to data measures from another day.

After the experiment is complete, we state to the program this fact (line 35). Then we save the state of the experiment, rename the statistical files with their original name followed by the current date at which it ended, inform the user of the finished experiment by displaying an image (named *finishedIcon*), close console outputs used for additional information, disallow the user from any operations by disabling buttons (line 39) and state to the system that no more operations are necessary. The only option left for the user after the whole process is to close the program.

5.8 USE CASES

In this section various diagrams which describe the behavior of the application are displayed. The order of cases displayed resembles that of section 5.7.5.

Use case	Starting the application
Actor	User
Preconditions	None.
Postconditions	All threads have been created and classes have been initialized.

Table 6: Use case "Starting the application"

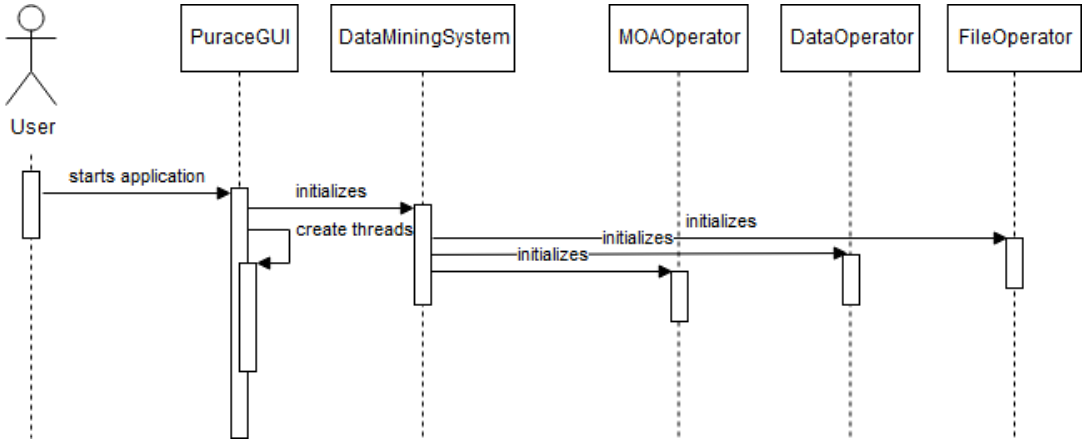

Figure 29: Diagram of use case "Starting the application"

Use case	Pressing experiment button
Actor	User
Preconditions	Application initialized. Experiment not launched or unpaused.
Postconditions	Graphical interface unit retrieves the action of the user, changes the state of the interface and shows changes on the screen.

Table 7: Use case "Pressing experiment button"

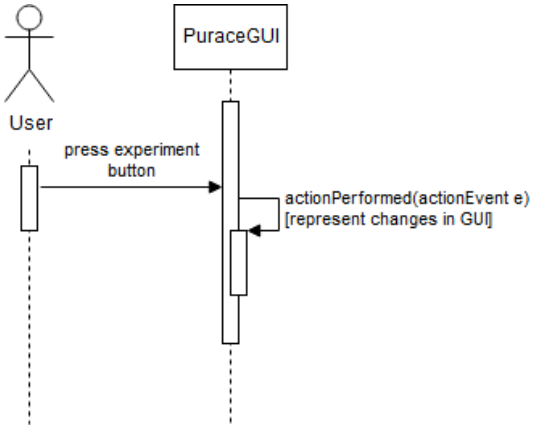

Figure 30: Diagram of use case "Pressing experiment button"

Use case	Pressing modification button
Actor	User
Preconditions	Application initialized. Experiment not launched or unpaused.
Postconditions	Graphical interface unit retrieves the action of the user, changes the state of the interface and shows changes on the screen.

Table 8: Use case "Pressing modification button"

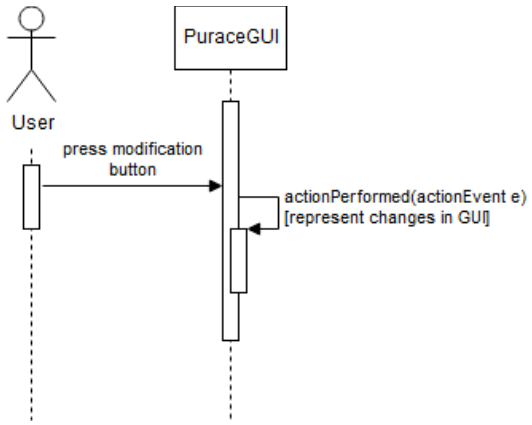

Figure 31: Diagram of use case "Pressing modification button"

Use case	Pressing calendar button
Actor	User
Preconditions	Application initialized. Experiment not launched or unpaused.
Postconditions	Graphical interface unit retrieves the action of the user, changes the state of the interface and shows changes on the screen.

Table 9: Use case "Pressing calendar button"

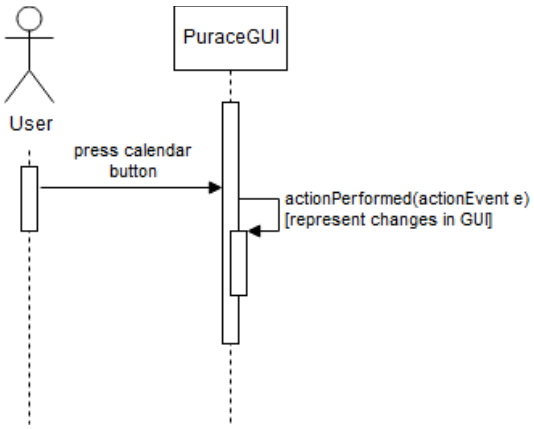

Figure 32: Diagram of use case "Pressing calendar button"

Use case	Launching a new experiment and training with one day of measures.
Actor	User
Preconditions	Application initialized. Experiment not launched or unpaused. Experiment selected with the first day of calendar chosen in "training" state.
Postconditions	Threads from GUI are run. Data mining system initializes variables and statistical files. Data mining system trains algorithms with files corresponding to one day of measures. Graphical interface unit retrieves the action of the user, changes the state of the interface and shows changes on the screen.

Table 10: Use case "Launching a new experiment and training with one day of measures"

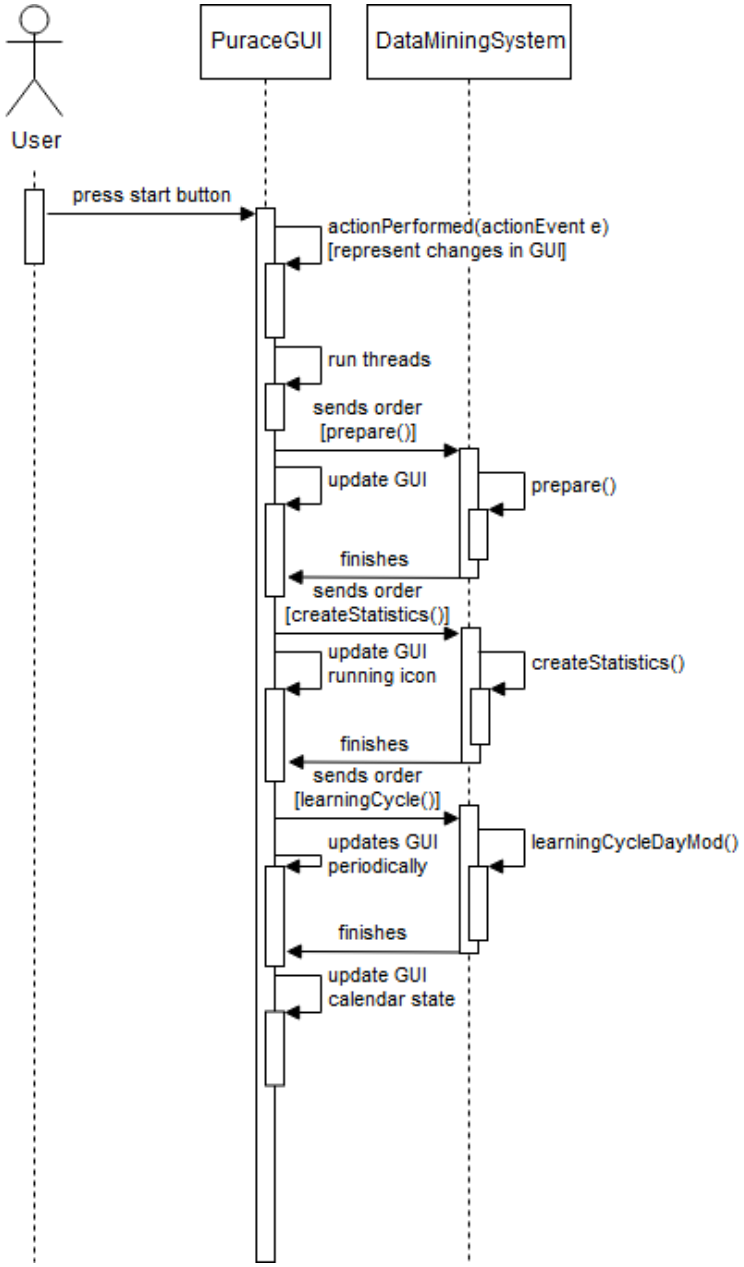

Figure 33: Diagram of use case "Launching a new experiment and training with one day of measures"

Use case	Launching an experiment, training with one day of measures and closing of application.
Actor	User
Preconditions	Application initialized. Experiment not launched or unpaused. "Start" button not pressed. Experiment selected with the first day of calendar chosen and not in "completion" state in "training" state.
Postconditions	Threads from GUI are run. State of experiment is saved. Data mining system initializes variables and statistical files. Data mining system trains algorithms with files corresponding to one day of measures. Graphical interface unit retrieves the action of the user, changes the state of the interface and shows changes on the screen. The application is closed.

Table 11: Use case "Launching an experiment, training with one day of measures and closing the application"

Use case	Loading a halted experiment, training with one day of measures and closing of application.
Actor	User
Preconditions	Application initialized. Experiment launched previously, save state correctly generated. "Start" button not pressed. Experiment selected with the first day of calendar chosen and not in "completion" state in "training" state.
Postconditions	Threads from GUI are run. State of experiment is saved. Data mining system uses already initialized statistical files. Data mining system trains algorithms with files corresponding to one day of measures. Graphical interface unit retrieves the action of the user, changes the state of the interface and shows changes on the screen. The application is closed.

Table 12: Use case "Loading a halted experiment, training with one day of measures and closing the application"

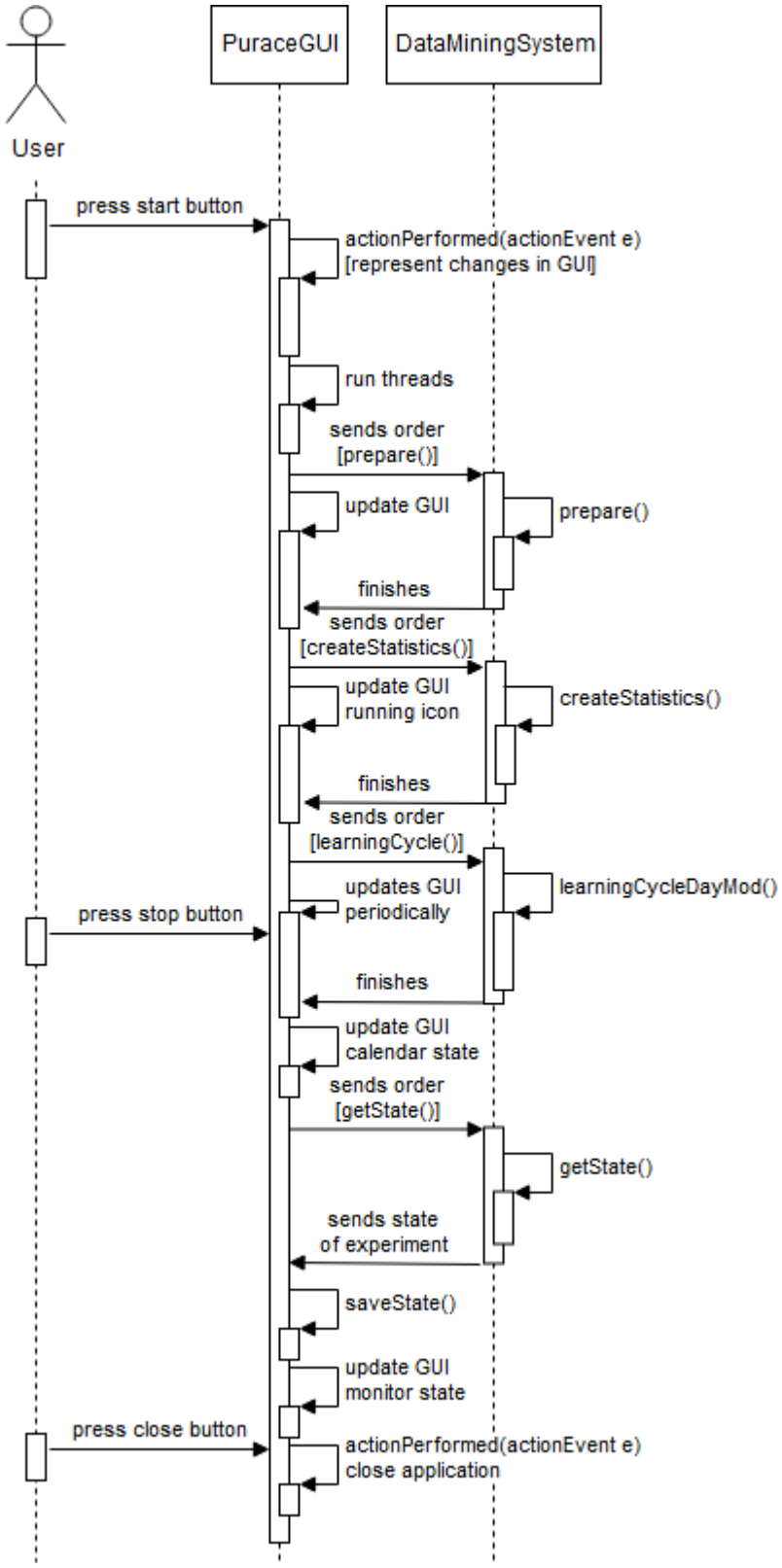

Figure 34: Diagram of use case "Launching an experiment, training with one day of measures and closing the application"

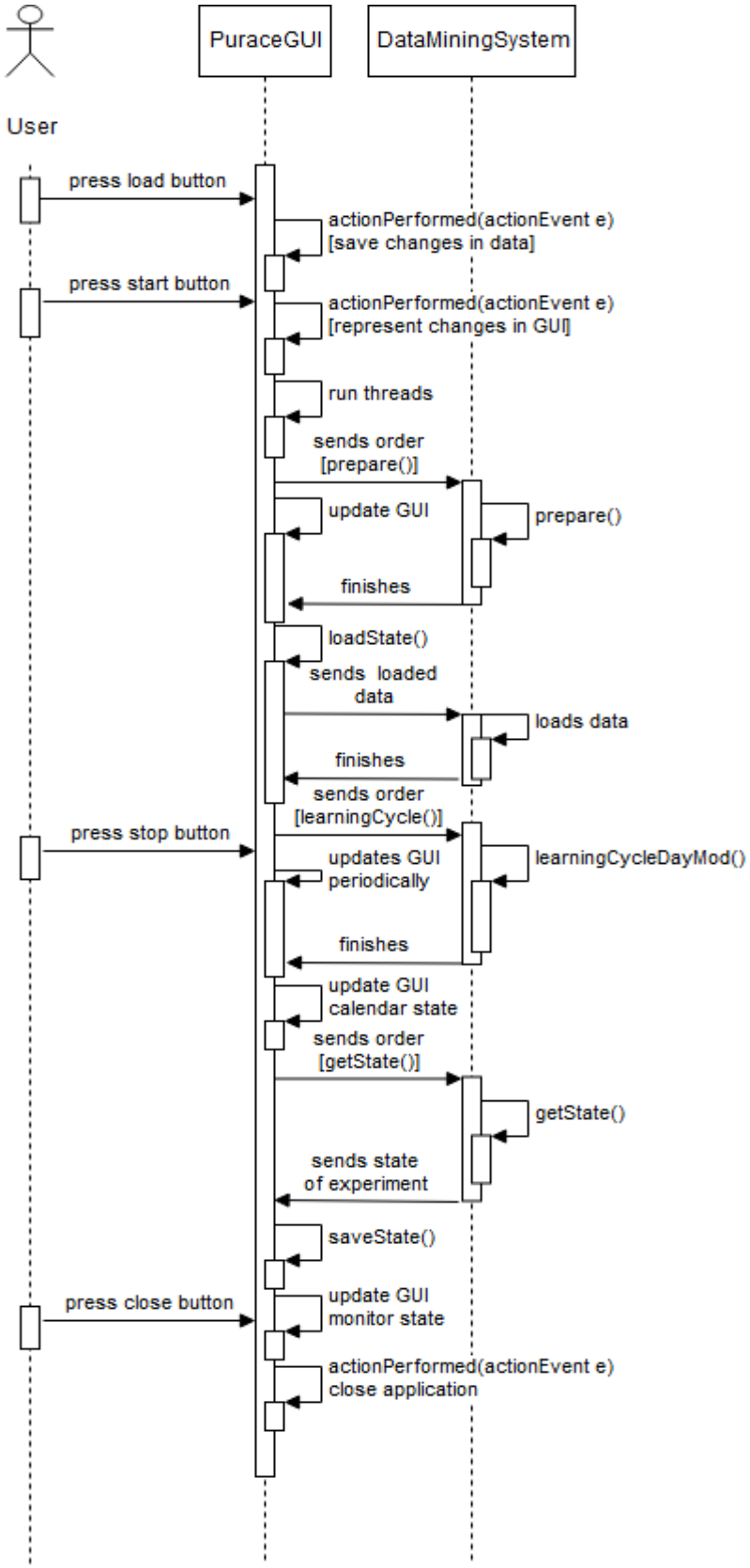

Figure 35: Diagram of use case "Loading of a halted experiment, training with one day of measures and closing the application"

Next chapter will cover the studies made with the application, the results obtained and the meaning behind these results.

6. EXPERIMENTATION

Now it's the time to put our application to use and see what we are able to find out. Going to the matter of options chosen for the tests performed, the early analysis of the problem made us take the decision of including in the application methods for undersampling. Undersampling is the process by which instances selected for the learning phase of algorithms are discarded before its inclusion.

What would be the reason to limit the quantity of data to learn for an algorithm? , the reader may ask. There are multiple reasons! Having repeated instances would be the first to come to mind, but the reasoning for this work follow two other causes: using a smaller amount of instances to save time and upgrade accuracy. When some classes contain a large amount of instances compared with the rest, algorithms can behave so that they give more priority to those classes and little priority to others. Just to give an example, imagine we wanted to correctly predict the result of flipping a tricked coin which most of the time falls in tails. The most efficient thing to do would be to pick always tails. And so, the same kind of behavior may occur with some algorithms.

On the subject of algorithms, most of the ones chosen will be based on the Hoeffding tree algorithm, so it will be taken as the initial reference to this decision. Also, since the second and third approaches are based on the premise that the first can give substantial accuracy, these initial tests will be using files generated from modification type 6 and variants of this modification.

6.1. USAGE OF UNDERSAMPLING

After training a Hoeffding tree using the first week of data without undersampling, which translates to the use of 2.519.993 instances, the results are the following:

Total accuracy	Total accuracy without including NULL class
0.9941772314689256	0.5466390321492264

Table 13: Results of training with one week of data measures using modification type 6 for algorithm Hoeffding tree

The reader may find strange the difference, but the number of "NULL" samples is quite big compared to the sum of the rest of the samples. Knowing from the data already given about the original instance files, we know that a week of data will have with this modification 60.479.993 modified instances. In this case the actual number of "NULL" samples is 59.961.318, leaving the samples of other classes combined with only 518.675 instances.

In cases where one class is represented by such a large division of the total, undersampling may not only give better accuracy to the prediction of other classes, but by using millions of samples less it will make our training of algorithms much faster.

To prove the possibilities of undersampling for this problem, it was decided to change proportion of "NULL" samples to 50% of the total and 20% of the total, repeating the same training.

Total accuracy	Total accuracy without including NULL class
0.6675008434954451	0.722037884995421

Table 14: Results of training with one week of data measures using modification type 6 for the algorithm Hoeffding tree with undersampling of the NULL class to 50% of the total

Total accuracy	Total accuracy without including NULL class
0.7231650530660468	0.7989974454137948

Table 15: Results of training with one week of data measures using modification type 6 for the algorithm Hoeffding tree with undersampling of the NULL class to 20% of the total

This proves that performing undersampling can indeed upgrade the prediction rate of other classes, at the cost of lowering the accuracy for class "NULL". At first, the reader may believe that if we were to choose between the two types of undersampling performed, next trainings should use the one corresponding to including "NULL" samples as 20% of the total, since both rates of accuracy are higher compared to the 50% inclusion. A more meticulous analysis will show otherwise:

Accuracy for NULL class	Total samples of NULL class
0.6129638019954692	518675

Table 16: Accuracy and number of samples for NULL class after training with one week of data measures using modification type 6 for Hoeffding tree with undersampling of the NULL class to 50% of the total

Accuracy for NULL class	Total samples of NULL class
0.41983372921615203	129668

Table 17: Accuracy and number of samples for NULL class after training with one week of data measures using modification type 6 for Hoeffding tree with undersampling of the NULL class to 20% of the total

A quick calculation will show 317.929 correct for the 50% undersampling case and only 54.440 correct for the other. Not only is the accuracy when performing 20% undersampling lower, but the number of instances is 75% less, making it less likely to adapt well to our necessities. Also, it should be pointed out that all "NULL" samples from the 20% undersampling performed are part of the samples from the 50% undersampling training.

With these results it was decided to use 50% undersampling, but not before trying other alternatives modifications of data before realizing this fact! This however won't be a problem to prove the usefulness of modification 6 as it is.

6.2. POSSIBILITIES OF MODIFICATIONS

This time the author will search for other ways to tackle the main approach, testing modifications not included in the final application. As stated before, 50% undersampling of the "NULL" class won't be used here and instead choose to perform 20% undersampling because of lack of knowledge about the accuracy comparison.

Using the same information from each instance of modified files using modification type 6, would there be another way to express the same data and get better results? It was decided to try what would happen if fusing the values of the five seismometers together creating the following structure, which will be named modification 6_sum:

x	Sum of X axis value of five seismometers
y	Sum of Y axis value of five seismometers
z	Sum of Z axis value of five seismometers
x_mov	Sum of X axis movement value of five seismometers
y_mov	Sum of Y axis movement value of five seismometers
z_mov	Sum of Z axis movement value of five seismometers
class	Seismic movement of the instance

Table 18: Structure of modification type 6_sum

We will need a way to compare fairly the results of using this structure for modified instances with results of using modification 6. The author will follow the same process as before, training Hoeffding tree algorithm with one week of data instances. The data chosen will include the same files as before.

Total accuracy	Total accuracy without including NULL class
0.4420098620637533	0.5106203306502145

Table 19: Results of training with one week of data measures using modification type 6_sum for Hoeffding tree with undersampling of the NULL class to 20% of the total

We would expect that, if we considered this structure an upgrade to the original of the modification, the same data would give us better or comparable results, but they are actually worse than those of not using undersampling. The reason for this result could have a very simple explanation. Seismometers are separated from each other by at least tens of meters. This means that sometimes, when part of a seismic wave reaches one of them first, the measured movement of other seismometers may have very little relation to the one specific of the first machine, with the exception of the possibility of belonging to the same class. This union may produce noise and lose important data in the process. The variation between the original modification and this one would be that with the original the same behavior may be seen for each seismometer, but with some time (or, in this case, instances) of difference.

A new question comes to mind: which would be more efficient, to use one seismometer with more data for each instance or instead multiple seismometers but little data? To solve it, a new modification type called 6_vec5 will be generated, using the same number of attributes as the original modification.

The structure of each modified data instance will be composed of five points and five vectors from measures taken by the ABLO seismometer with the class they correspond to. Recalling the explanations and figures 4 to 6 in the section 5.3 (Modification types), the representation of part of 3 instances using one axis of the seismometer will be included next, with a few differences. This time we will be able to show one third of each of these instances, since its structure only uses data from 3 axes of measures taken by a single seismometer. A figure with three instances in a sequence is included, so that the reader may understand the order followed by each of the instances:

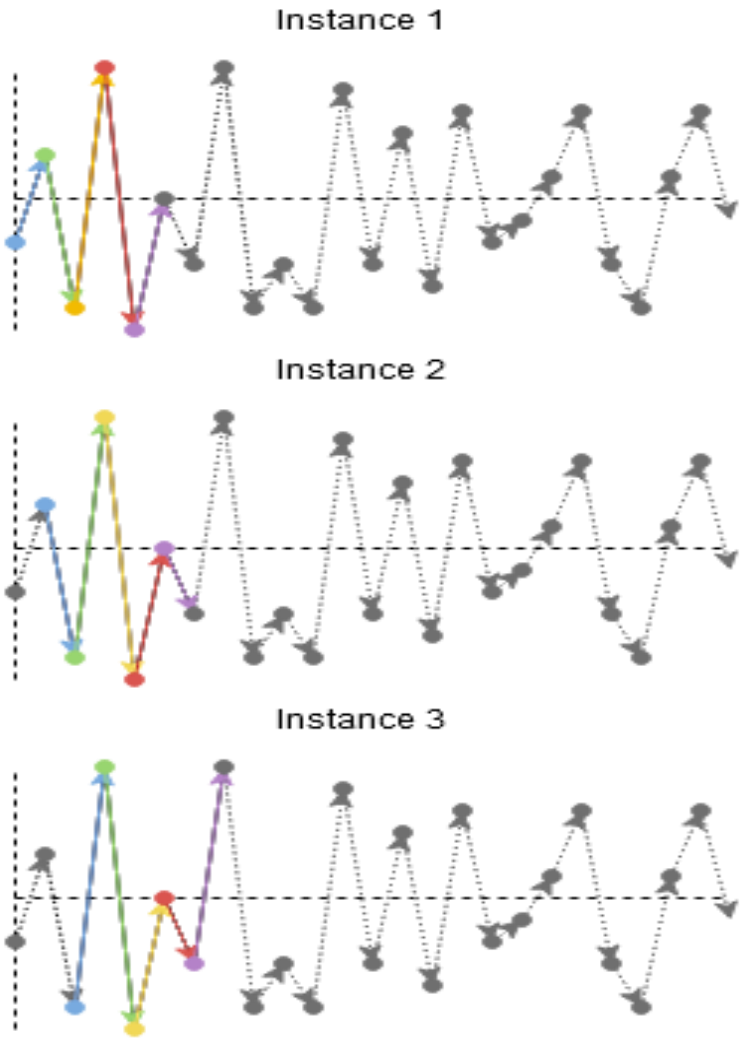

Figure 36: Representation example of part of the structure given by 3 data instances modified by scheme 6_vec5 with the image of an axis of a seismometer

In figure 29 we can differentiate vectors with dotted lines and vectors with full lines. There are five vectors with full lines for each of the instances and they are colored depending on their order, following next scheme: blue, green, yellow, red and purple. This structure is followed for every instance included in modified files when using this type of modification. The reader may realize that, if we apply the same mechanisms as before for instances including one or more vectors, and if we don't take into account the first 4 and the last 4 points and vectors of a file of data instances for a day of measures, all points and vectors will be each included in 5 different instances.

Could this structure help us predict events more accurately? To answer this and make a fair comparison, some of the characteristics tweaked for this modification must be determined first. The class of the seismic movement will be defined by the one to which the first point measured from the instance belongs to. Undersampling will be performed for the "NULL" class to make their samples a 20% of the total used for training. Including now more than one original instance to create modified ones, this means that to save time for the same reasons as before (need for multiple files of large size), 8.639.995 instances from each instance file will be

used, corresponding to a day of measures. The same week of data used for the other experiments will train a previously chosen algorithm, which will again be Hoeffding tree.

To have a better understanding of the composition of the instances using this kind of modification, its structure will be defined here:

ABLO_x1	X axis of a 1st instance from seismometer ABL
ABLO_y1	Y axis of a 1st instance from seismometer ABL
ABLO_z1	Z axis of a 1st instance from seismometer ABL
ABLO_x2	X axis of a 2nd instance from seismometer ABL
ABLO_y2	Y axis of a 2nd instance from seismometer ABL
ABLO_z2	Z axis of a 2nd instance from seismometer ABL
ABLO_x3	X axis of a 3rd instance from seismometer ABL
ABLO_y3	Y axis of a 3rd instance from seismometer ABL
ABLO_z3	Z axis of a 3rd instance from seismometer ABL
ABLO_x4	X axis of a 4th instance from seismometer ABL
ABLO_y4	Y axis of a 4th instance from seismometer ABL
ABLO_z4	Z axis of a 4th instance from seismometer ABL
ABLO_x5	X axis of a 5th instance from seismometer ABL
ABLO_y5	Y axis of a 5th instance from seismometer ABL
ABLO_z5	Z axis of a 5th instance from seismometer ABL
ABLO_x1_move	Movement of X axis of a 1st instance from seismometer ABL
ABLO_y1_move	Movement of Y axis of a 1st instance from seismometer ABL
ABLO_z1_move	Movement of Z axis of a 1st instance from seismometer ABL
ABLO_x2_move	Movement of X axis of a 2nd instance from seismometer ABL
ABLO_y2_move	Movement of Y axis of a 2nd instance from seismometer ABL
ABLO_z2_move	Movement of Z axis of a 2nd instance from seismometer ABL
ABLO_x3_move	Movement of X axis of a 3rd instance from seismometer ABL
ABLO_y3_move	Movement of Y axis of a 3rd instance from seismometer ABL
ABLO_z3_move	Movement of Z axis of a 3rd instance from seismometer ABL
ABLO_x4_move	Movement of X axis of a 4th instance from seismometer ABL
ABLO_y4_move	Movement of Y axis of a 4th instance from seismometer ABL
ABLO_z4_move	Movement of Z axis of a 4th instance from seismometer ABL
ABLO_x5_move	Movement of X axis of a 5th instance from seismometer ABL
ABLO_y5_move	Movement of Y axis of a 5th instance from seismometer ABL
ABLO_z5_move	Movement of Z axis of a 5th instance from seismometer ABL
class	Seismic movement of the instance

Table 20: Necessary attributes of a data instance for files using modification type 6_vec5

The results of training are not as good as expected, but quite comparable to the ones form modification type 6_sum:

Total accuracy	Total accuracy without including NULL class
0.44253273344510546	0.507394804068058

Table 21: Results of training with one week of data measures using modification type 6_vec5 for Hoeffding tree with undersampling of the NULL class to 20% of the total

The original modification is still more accurate by a margin of at least 22%. The author of this work won't discourage however, finding other ways to obtain better results by adding new data. On the contrary, he believes that being able to use the vectors of movement of five seismometers using a sequence of original instances could give better results, but the limitations of this work can't make it possible to include that study here. Since we are searching for efficiency, these experiments prove original modification type 6 is the best alternative, and can be used as base for creating modifications which will be of interest for next approaches.

6.3. EXPERIMENTS WITH MODIFICATION TYPE 6

After proving modification type 6 with undersampling of "NULL" instances to 50% of the total to be the most efficient of the ones reviewed, it was decided to train as many algorithms as possible from the MOA library, with as much data as possible. This is of course, taking into account the limitations of time for this work. The author was able to use the data from the first five months available. In practice this translated into checking 153 days of data, which is the equivalent to 1.321.919.847 instances. From these instances, the ones included in the training are a total of 88.863.258 instances, with approximately half of these of "NULL" class. Comparing the proportion of "NULL" samples to the whole set of data, it was found that their samples compose 96,6388% of the total. The set of instances from other classes is quite small, composing only 3,3611%. The proportion of classes within this set can be found in figure 30:

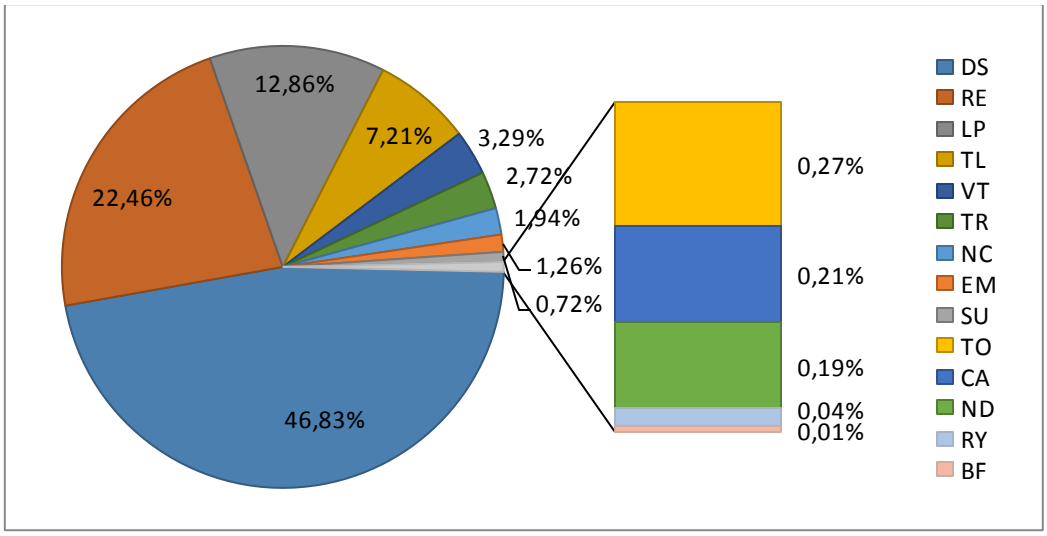

Figure 37: Proportion of samples used for training with modification type 6 divided by class, from files which include measures from 1st of August 2015 to 31st of December 2015 without counting NULL samples

We can see the number of training instances by class in table 15:

VT samples	1462649
LP samples	5712597
HB samples	0
HI samples	0
TO samples	120225
TL samples	3201602
RE samples	9977279
DS samples	20805999
TR samples	1209810
EM samples	559019
SU samples	321297
NC samples	860179
CA samples	94340
RY samples	18214
EX samples	0
AV samples	0
BF samples	6210
ND samples	82209
NULL samples	44431629

Table 22: Number of samples divided by class from files which include measures from 1st of August 2015 to 31st of December 2015

When we talked about data being insufficient for the objectives proposed, it referred mainly to the lack of instances for some of the classes and the low number of instances for others. This unbalanced proportion may be disadvantageous for the models produced by algorithms.

Making the distinction from total instances and set of instances without those belonging to "NULL" class is fundamental. While it's important to predict all kinds of events with the greatest accuracy possible, our interests lie in correctly finding the events that may suppose some kind of damage to the population or the calibration of the array of seismometers (examples of last kind of events belong to instances from "CA" or "Calibration" class). The reader will find that he/she will be more accurate than the algorithms used by just saying that all instances belong to "NULL" class, but that doesn't mean it will be more useful than the algorithms used.

Following, is the list of algorithms tried and the accuracy of the model produced by their training from five months of data:

Algorithm	Total accuracy	Total accuracy without including NULL class
NaiveBayes	0.0724315442046925	0.11116920786316432
SGD	0.0806203279200049	0.16124065584000982
MajorityClass	0.4999999549870206	0.000000450129794
ARFHoeffdingTree	0.5687560993993716	0.3240452201291112
HoeffdingTree	0.6209812046279014	0.5971969877584277
AdaHoeffdingOptionTree	0.8702711417580481	0.8584825688025077
SGDMultiClass	0.9979591452746421	0.9975382851706832

Table 23: Accuracy of the training of different algorithms using five months of data measures with modification type 6 and undersampling of the NULL class to 50% of the total

Given undersampling, it is expected to get lower accuracy when using Bayesian networks. Still, Hoeffding tree was proven to be more accurate when undersampling wasn't performed in variations of first experiment. From table 16 we could affirm that using a multiple class stochastic gradient descent would be the best solution, but this was a process of supervised learning. Reality will be "unsupervised" since we won't have the correct class of an instance unless we correctly guess it or an expert confirms it. We will need to test the models created and compare their behavior with new data.

In a realistic setting, if we wanted to minimize the difference between the date at which last instances of data were retrieved and the date of first usage of a prediction system (useful for some algorithms like variants of SGD), given the parameters for the application, this difference would be expected to be close to 24 hours. In this time, the experts classifying the instances would be able to finish classification of last 24 hours of measures, since it would be impossible for an expert to classify instances within a time frame as small as 10 milliseconds. The author assumes that writing and performing the correct classification of 1 hour of measures from seismometers by an expert to take at the very least 1 minute. This is however, without taking into account the time an algorithm would take to learn from these instances and setting the correct disposition of the model for real-time prediction. Since the system is prepared to only analyze full days of measures (although small modifications in design permit to change this easily), the author has to agree on the difference chosen. This can also justify not using data from for the 1st of January of 2016 given the lack of measures from seismometer "LAR" on this day, which could produce an error in the system (a fixable error, but using the data could also add noise to the results).

We will use then for testing, the data captured from the 2nd to the 11th of January 2016. The amount of instances will be 86.399.990, close to the number of instances used for models. Since no undersampling is performed for testing, all "NULL" class instances contained in the data will be used this time. The proportion of each of the classes will vary, and 3 of the classes with least instances (BF, RY & CA) will not appear, as we can see in figure 31:

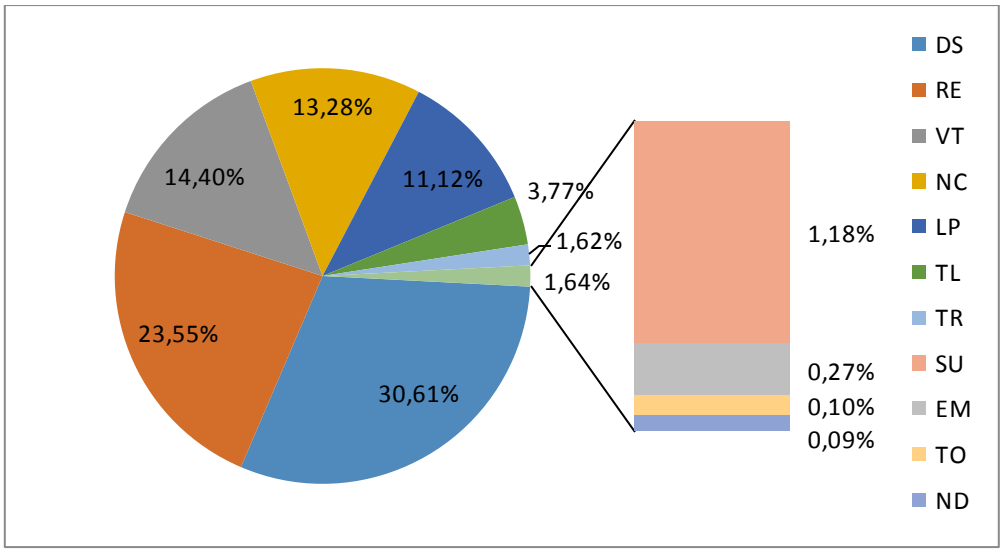

Figure 38: Proportion of samples used for testing with modification type 6 divided by class, using files which include measures from 2nd to 11th of January 2016

We can see the number of test instances by class in table 17:

VT samples	835528
LP samples	645250
HB samples	0
HI samples	0
TO samples	5684
TL samples	218871
RE samples	1366124
DS samples	1775622
TR samples	93969
EM samples	15757
SU samples	69176
NC samples	770120
CA samples	0
RY samples	0
EX samples	0
AV samples	0
BF samples	0
ND samples	5393
NULL samples	80599496

Table 24: Number of samples divided by class from files which include measures from the 2nd to the 11th of January 2016

The results of applying the models given by training previous algorithms to the subset of data used for testing are the following:

Algorithm	Total accuracy	Total accuracy without including NULL class
NaiveBayes	0.01040262	0.14590912
SGD	0.00746817	0.11124053
MajorityClass	0.93286465	0
HoeffdingTree	0.34432387	0.09061073
AdaHoeffdingOptionTree	0.59256723	0.05232261
ARFHoeffdingTree	0.7031247	0.05567957
SGDMultiClass	0.02021823	0.30061405

Table 25: Accuracy of testing for different algorithms using measures from 2nd to 11th of January 2016 modified with modification scheme 6

Unexpectedly, the algorithm SGDMulticlass reached one of the worst accuracy levels, but the best one if we look at accuracy without instances belonging to "NULL" class. On the other hand, the promising AdaHoeffdingOptionTree was surpassed by ARFHoeffdingTree, giving us a model quite reliable for the instances of the "NULL" class given they are the majority. Using a higher number of instances for some classes to balance proportion would be beneficial.

	VT	LP	HB	HI	TO	TL	RE	DS	TR	EM	SU	NC	CA	RY	EX	AV	BF	ND	NULL
VT	16655	0	0	0	0	0	13	818860	0	0	0	0	0	0	0	0	0	0	0
LP	5891	0	0	0	0	0	0	639353	0	0	0	0	0	0	0	0	0	0	6
HB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TO	0	0	0	0	0	0	0	5684	0	0	0	0	0	0	0	0	0	0	0
TL	2381	12	0	0	7	0	0	216448	0	0	0	0	0	0	0	0	0	0	23
RE	15430	0	0	0	0	0	0	1350694	0	0	0	0	0	0	0	0	0	0	0
DS	48567	0	0	0	0	0	0	1727055	0	0	0	0	0	0	0	0	0	0	0
TR	4186	0	0	0	0	0	0	89783	0	0	0	0	0	0	0	0	0	0	0
EM	0	0	0	0	0	0	0	15757	0	0	0	0	0	0	0	0	0	0	0
SU	0	0	0	0	0	0	0	68176	0	0	0	0	0	0	0	0	0	0	0
NC	6965	175	0	0	75	55	18	762760	0	0	0	0	0	0	0	0	0	0	72
CA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EX	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AV	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ND	0	0	0	0	0	0	0	5393	0	0	0	0	0	0	0	0	0	0	0
NULL	2632620	7485	578	0	1128	1857	1446	77951211	0	1	0	1	3	0	0	0	5	16	3145

Table 26: Confusion table of algorithm SGDMulticlass from testing using modification type 6

The reader may wonder about the reason why the accuracy of SGDMulticlass for classes different than "NULL" is so low compared to the results for training. Inside the confusion matrix we can appreciate a clue which may give us the reason. Most of the time the algorithm predicts the event to which instances belong to be of class "DS". We can attribute then the result to three factors:

-The class "DS" is the second most common in the training set, making more likely to find instances of this class.

-The characteristics of the class "DS" might be more distinguishable than those of the "NULL" class since the second englobes data for which we would lack interest because their instances don't indicate any kind of danger (at least without further knowledge).

-Since the algorithm is stochastic and depends on recently analyzed data, and in this kind of testing we don't learn from instances, it loses data which conditions the response for next query.

The algorithm that gives the most promising result comparing training accuracy and testing accuracy is actually AdaHoeffdingOptionTree, although it must be stated that ARFHoeffdingTree gave better results while relying in a much lower accuracy.

	VT	LP	HB	HI	TO	TL	RE	DS	TR	EM	SU	NC	CA	RY	EX	AV	BF	ND	NULL
VT	249613	15354	0	0	0	15168	13374	15877	978	284	13	0	0	0	0	0	0	0	524867
LP	191976	14586	0	0	0	13999	10559	9495	331	40	18	0	0	0	0	0	0	0	404246
HB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TO	885	2	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	4795
TL	62148	5576	0	0	0	2478	7420	2862	405	0	50	1	0	0	0	0	0	0	137931
RE	463174	29951	0	0	0	34324	10534	23049	2928	20	35	0	0	0	0	0	0	0	802109
DS	589938	18303	0	0	0	25744	10914	26184	352	21	39	0	0	0	0	0	0	0	1104127
TR	36464	447	0	0	0	2703	27	2796	3	209	0	0	0	0	0	0	0	0	51320
EM	4407	67	0	0	0	0	599	87	0	0	0	0	0	0	0	0	0	0	10597
SU	18195	518	0	0	0	2600	3266	650	108	1	98	0	0	0	0	0	0	0	42740
NC	233561	6365	0	0	0	14042	10856	10199	752	130	1	1	0	0	0	0	0	0	494213
CA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EX	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AV	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ND	1998	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3359
NULL	23833957	1371283	0	0	0	1729421	1326307	1297262	103418	37593	5700	249	0	0	0	0	0	0	50894306

Table 27: Confusion table of algorithm AdaHoeffdingOptionTree from testing using modification type 6

We can observe in table 20 that the algorithm AdaHoeffdingOptionTree has a wider set of predictions, and the higher the amount of instances trained with, the higher the accuracy for the class.

The last question for this experiment is whether a real time application could be created from these algorithms, or putting it in another way, if the time we spend learning and testing these algorithms is low enough to make the difference between a query and an answer smaller than or comparable to the time it takes to create an instance. We can compare the mean time expended for training and testing of AdaHoeffdingOptionTree and SGDMulticlass with next table:

Algorithm	Instance training mean time (ms)	Instance testing mean time (ms)
AdaHoeffdingOptionTree	129.066176	58.2916667
SGDMulticlass	1.71541394	1

Table 28: Mean time for training and testing of an instance for algorithms AdaHoeffdingOptionTree and SGDMulticlass using modification type 6

For the training of algorithms we will have available all instances classified previously, and we won't have to wait for each individual instance to be used, so the times given are acceptable.

Let's remember that the difference between 2 instances produced is of 10 milliseconds. Since the mean times are quite lower, using the algorithm SGDMulticlass would be acceptable if a higher volume of data supposed a greater accuracy, even if we had to train this algorithm with the predictions of class given for each instance to ensure not tampering the stochastic element of it.

The algorithm AdaHoeffingOptionTree however, can't be used by itself to create a real-time categorization, since it takes longer to do so than it takes the seismometers to produce various instances. Still, there is a solution for this problem. Let's suppose the time for testing an instance to be always of 60 milliseconds. If we used an array of 6 copies of the same model produced by the algorithm to analyze every group of 6 instances concatenated (that is, one instance for each of the models) we could have a system that produces the categorization of instances with a latency of 60 milliseconds, which is a sufficiently small time frame to still be useful as a real-time application.

6.4. EXPERIMENTS WITH MODIFICATION TYPE 7

With modification type 7 the author hopes to predict the next seismic event that will occur from any point in time (with the exceptions made in section 4.2). This modification will be built over the structure of the last, meaning that this prediction will be done after classification by models using data modified by modification type 6 by adding a field to these instances (the next seismic event occurring). Given the structure of these new data instances, all "NULL" class instances will be the ones actually used for prediction, associating them with the class of next seismic event as preamble of that event.

Since there won't be "NULL" class samples, no undersampling will be performed, meaning the number of instances used will be the maximum for each day of data (8.639.999 instances).

Sadly for the curiosity of the reader, the author of this work didn't have enough time to use 5 months of measures for the training of algorithms. However, he was able to use the first 20 days of August 2015 for training and the next 10 days for testing of algorithm AdaHoeffdingOptionTree, which presented the best balance between training and test accuracy rates. This may seem like a smaller test than the one for modification type 6, but it actually translates to using 172.799.980 instances for training and 86.399.990 for testing, which makes for a bigger training sample and an equal size test sample.

Focusing on the training sample, the proportion of classes within this set can be found in figure 32:

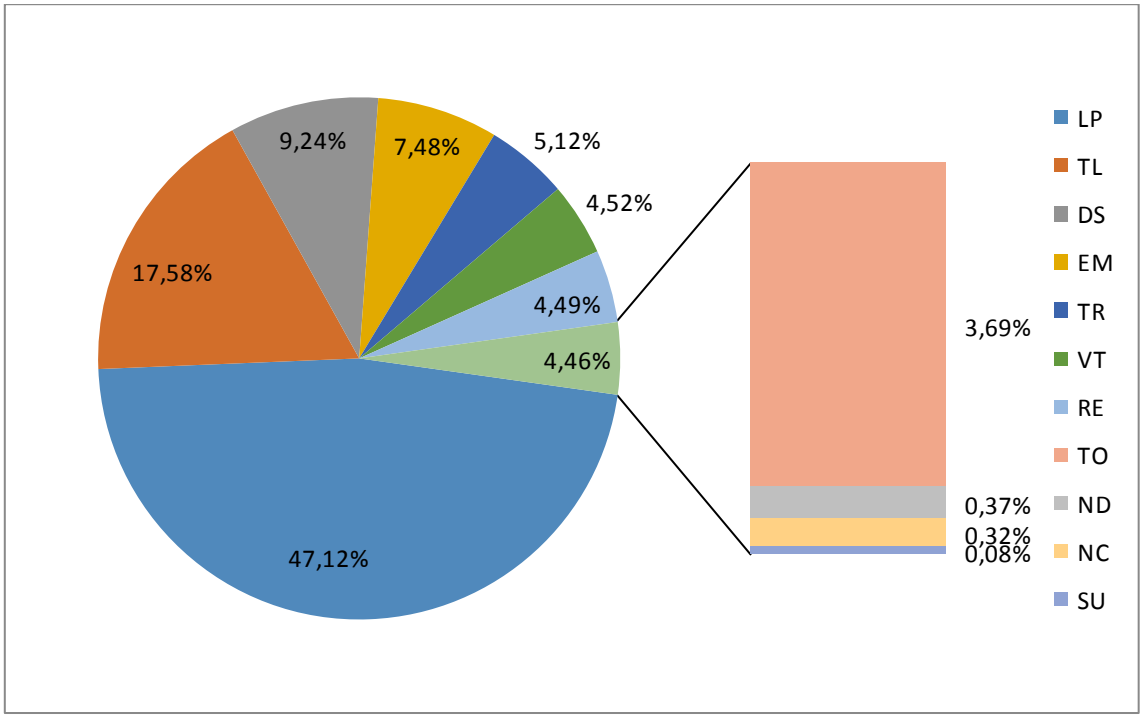

Figure 39: Proportion of samples used for training with modification type 7 divided by class, from files which include measures from 1st of August to 20th of August 2015

This time we won't need to difference between accuracy with or without "NULL" class, since we are predicting the next seismic event that will occur meaning no "NULL" class instances exist for this modification. The result from training is:

Algorithm	Total accuracy
AdaHoeffdingOptionTree	0.8828893672325656

Table 29: Accuracy of algorithm AdaHoeffdingOptionTree for training using modification type 7

Compared to the accuracy of last training process, the accuracy rates are similar, with the result of modification type 7 being slightly better. Will this rate maintain for testing? Before showing the answer let's compare the proportion of samples by class of the training set with the one from the testing set.

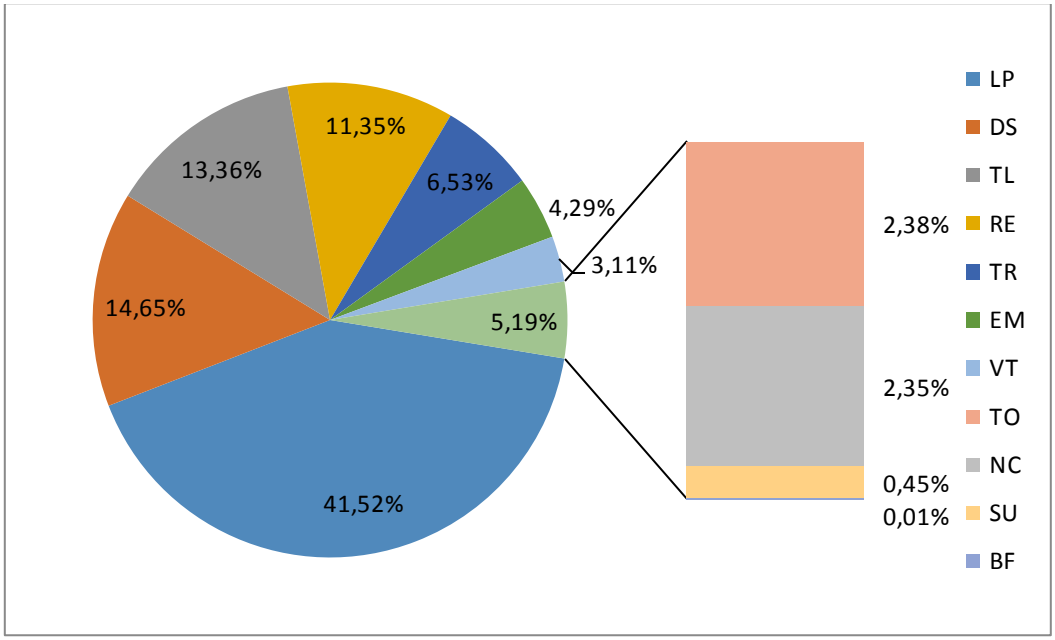

Figure 40: Proportion of samples used for testing with modification type 7 divided by class, from files which include measures from 21st of August to 30th of August 2015

In this case there aren't any instances of class "ND" and a very small number of instances of "BF" class. Since no "BF" class instances appeared in the training set, the algorithm won't be able to correctly classify these instances. Some of the classes which had a greater proportion of instances in the training set have now a much smaller proportion in the testing set and vice versa. Still, compared to training set, most of the instances in testing belong to "TL" class. The result from testing is:

Algorithm	Total accuracy
AdaHoeffdingOptionTree	0.24179973

Table 30: Accuracy of algorithm AdaHoeffdingOptionTree for testing using modification type 7

While giving a low accuracy result, it is still much higher than the accuracy from testing using modification type 6 not containing "NULL" class samples.

In this case, using more fields for instances than those of last modification scheme will be more time consuming for algorithms, given instances produced from this modification are composed over the original structure from instances produced from modification type 6.

Algorithm	Instance training mean time (ms)	Instance testing mean time (ms)
AdaHoeffdingOptionTree	165.99375	81.3916667

Table 31: Mean time for training and testing of an instance for algorithm AdaHoeffdingOptionTree using modification type 7

Since the models are predicting instead of categorizing, a higher testing time can be forgiven for events that last longer than the testing mean time.

Looking at the confusion table we will find that predictions are spread between classes.

	VT	LP	HB	HI	TO	TL	RE	DS	TR	EM	SU	NC	CA	RY	EX	AV	BF	ND
VT	566793	1141868	0	0	46757	800443	263741	444075	277567	99155	397	14274	0	0	0	0	0	1000
LP	2669053	15433494	0	0	320178	7623030	1509618	2990117	3911020	911903	5180	53533	0	0	0	0	0	34838
HB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TO	446739	756016	0	0	82383	358993	87377	81067	97848	118755	374	1902	0	0	0	0	0	517
TL	1300088	4311600	0	0	202769	1762596	409909	1247451	1739566	333046	3666	12606	0	0	0	0	0	81964
RE	944792	3935370	0	0	72158	1468827	704922	885569	1341230	298726	4346	14895	0	0	0	0	0	23896
DS	1211315	4782217	0	0	62024	2217153	262197	1307221	1897073	637374	13957	16116	0	0	0	0	0	105106
TR	686889	1972048	0	0	302597	1265250	204728	342295	635859	132045	1168	28526	0	0	0	0	0	4404
EM	683690	1361145	0	0	95602	569378	91798	144005	302194	383469	1401	1319	0	0	0	0	0	29270
SU	50370	160786	0	0	13225	49012	3997	57051	41184	2545	304	2638	0	0	0	0	0	25
NC	104512	682896	0	0	21931	428084	55851	399253	208607	89109	46	14453	0	0	0	0	0	5107
CA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EX	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AV	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BF	5819	790	0	0	37	46	37	6	1326	113	0	0	0	0	0	0	0	0
ND	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 32: Confusion table of algorithm AdaHoeffdingTree from testing using modification type 7

Could this result mean classifying "NULL" class instances as preambles of the events that arise right after them is a viable solution for predictions? Could this also be the answer to getting rid of "NULL" class samples? Sadly, this data doesn't prove yet the veracity of other more accurate models based on the same assumptions that could be tested. Further studies will be necessary to validate the scope of this experiment.

6.5. EXPERIMENTS WITH MODIFICATION TYPE 8

This modification type takes the objective given by third approach. By dividing seismic events in 16 equal parts we can predict the total length of these events from as few instances as possible. All "NULL" class instances are classified with the same value, making it easy to distinguish these instances from the ones from other classes. Since instances with a class equivalent to the "NULL" class are being included, undersampling of 50% of the total samples will be performed.

For this experiment the author was able to use the same sets of data files as the ones from experiments with modification type 6. This means using the working with the same original data as with that experiment. Strangely enough, a much greater amount of instances different from class "-1" or "NULL" was found this time, which the reader would think impossible given the constraints of the modification scheme. As it turns out, a mistake in the code of this

modification scheme produced the misclassification of instances, which invalidated the tests run with this scheme. For this reason, this idea will be left as a suggestion for future works.

7. CONCLUSIONS

Categorization of seismic events is a complex matter. What we have seen in this work is just a starting step from the point of view of data mining. The author simplified the information retrieved to the very simple schemes, and analyzed it to found models that have low accuracy. This may seem unrewarding, but if we focus in the small size of information given to find the category of data, it would actually be quite difficult for experts to accurately find the class to which this data belongs, and yet, these models seem to find ways of doing so. This calls for further research, for which the author has shown the necessity for a bigger set of data and the study of new relations that may lie hidden inside these measurements.

Two secondary approaches were presented, hoping to instill curiosity for the matter at hand, and while doing so, the author was able to impart ways to construct a system able to generate models of categorization and prediction. It has also been shown the usefulness of MOA [16] as a library for data mining algorithms; a tool which without it would have made this a much harder task.

Finally, the author hopes this work helped programmers of data mining applications to learn from mistakes and advices included, and generate with that knowledge better solutions to the problems found in their future endeavors.

8. FUTURE WORKS

If the author decided to continue this line of work, the next step would usually be the training of algorithms for each individual seismic event. These algorithms would only differentiate between those instances belonging or not belonging to one specified class. By union of models per event we could generate a better system for prediction.

One important factor to upgrade from the included experiments is undersampling; the author didn't refine enough the rate used for the "NULL" class or included the possibility of oversampling those classes with less instances.

Although this document focused on the very basics of the domain given by the data, some of the qualities of seismic events included in the original class files could be useful for prediction, most likely those which can be correctly computed or approximated with fewest data samples.

If we think about how the instances of each file are classified and how seismic events are registered, we should find that most of the time instances of the same class form "chunks" together, making most likely to find an instance of one class with the next instance being also from the same class. We could take advantage of this fact if we found a way to express that quality inside the data, without forgetting that knowing when such events start and finish is what gives this line of thought a possibility of success.

Lastly, a study by Stephen R. Mcnutt from California Department of Conservation [18] revealed that "most volcanic tremors occur within 10 days of eruptions". In that case, there could be comparable factors hardly found by the naked eye inside movements with days of difference. This could be the precedent of a prediction system produced by comparing pairs of data instances with a fixed or variable time difference in between.

9. BUDGET

In this section it is provided the estimation about costs produced to create this work. This includes the tools used, the time expended and the machines that run Purace application and experiments, as well as the salary for the person that created this work. It is important to remind the reader that the values proposed are approximations, and may or may not fit the budget of similar works depending on some factors.

The author estimates that over at least 10 years in Spain, the salaries of entry positions for consulting and programming job offers have stagnated to around 22.000€ a year for 1800 hours of work. Given this estimation, the following budget can be expected for human costs:

Role	Time expended (hours)	Cost (€)
Consultant	220	2688.88

Table 33: Budget for human costs

Since the tools used either came with the equipment bought or were free, this part of the budget would amount to zero. Still, the equipment used for the experiments required the following budget:

Equipment	Time expended (hours)	Cost (€)
Computer with 4GB of RAM and 2.5GHz processor with 4 cores	168	500
Servers Isis and Juno from University Carlos III of Madrid	504	4000

Table 34: Budget for equipment used for experiments

The total amount of time and money that this project has supposed is the following:

	Cost (€)	Time (hours)
Human costs	2688.88	220
Equipment	4500	672
Total	7188.88	892

Table 35: Budget for the project

The total time expended in human costs is estimated as the minimal, and not the mean expected for this work. The same occurs with costs related to equipment, which may vary greatly depending on the modifications chosen for data and the algorithms chosen, as proven by section 6 (Experimentation). Note that the amount of time for experiments is large compared to that of human costs, meaning that replicating this in a real work environment would translate into having the people in charge of the project idling when experiments occur. This time can be used for other tasks like documentation of the project or development of different projects.

10. SOCIOECONOMIC IMPACT

This work shows the viability of initial approaches to solve the problem of categorizing seismic events and opening lines of investigation for secondary problems such as predicting seismic events and predicting the length of said events. Since the accuracy of these models is not sufficiently high, the processes explained in this work can't be used by themselves for the intentions they were conceived, but can give researchers useful information about this task.

The architecture, the design and implementation of the application presented can be used for systems of online data mining, as it has been the case of this work. Various problems that arise from the creation of this kind of applications were mentioned, and some of the answers which solve them were explained and tested. This could help the developing process of other data mining applications to be faster and less prone to faults.

This work offers the results of using various algorithms which could in the future, by using modifications generated from the data included and, by following the author's assumptions, decisions or/and facts proven, serve as mechanisms to prevent seismic events and the damage that this events can provoke to population.

11. BIBLIOGRAPHY

[1] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/2014_Mount_Ontake_eruption. [Accessed 29 August 2018].

[2] J. Eduardo Gómez, D. Camilo Corrales, J. C. Corrales, A. Sanchis, A. Ledezma and J. A. Iglesias, "Monitoring of vulcano Puracé through seismic signals: Description of a real dataset.," *Annals of geophysics = Annali di geofisica,* p. 6, 2017.

[3] M. d. l. M. Crespo Jiménez, "Clasificación de ondas sísmicas con técnicas de minería de datos," Madrid, 2017.

[4] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Seismology. [Accessed 24 September 2018].

[5] A. Ben-Menahem, "A Concise History of Mainstream Seismology: Origins, Legacy, and Perspectives," *Bulletin of the Seismological Society of America,* vol. 85, no. 4, pp. 1202-1225, 1995.

[6] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Dirty_thunderstorm. [Accessed 24 September 2018].

[7] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI magazine,* pp. 37-54, 1996.

[8] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Data_mining. [Accessed 22 September 2018].

[9] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," *ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence,* pp. 357-368, 2016.

[10] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Incremental_learning. [Accessed 18 September 2018].

[11] G. Piatetsky, "KDNuggets," [Online]. Available: https://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html. [Accessed 18 September 2018].

[12] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer and R. Wirth, "CRISP-DM 1.0," 2000.

[13] Just Great Software Co. Ltd., "https://www.just-great-software.com/," [Online].

[14] B. Moolenaar, "http://www.vim.org/," [Online].

[15] C. Campbell, "http://www.vim.org/scripts/script.php?script_id=1506," [Online].

[16] A. Bifet, G. Holmes, R. Kirkby and B. Pfahringe, "MOA: Massive Online Analysis," *Journal of Machine Learning Research 11,* pp. 1601-1604, 2010.

[17] T. U. o. Waikato, "https://www.cs.waikato.ac.nz/ml/weka/arff.html," 1 April 2002. [Online]. [Accessed May 2018].

[18] S. R. McNutt, "Volcanic Tremor," in *Encyclopedia of Earth System Science, Volume 4*, California, Academic Press, Inc., 1992, pp. 417-425.